Thermal Sensor Allocation and Placement for Reconfigurable Systems

Rajarshi Mukherjee[¶], Somsubhra Mondal and Seda Ogrenci Memik
[¶]Synopsys, Inc. Mountain View, CA 94043
Department of Electrical Engineering and Computer Science
Northwestern University, Evanston, IL 60208

ABSTRACT

Temperature monitoring using thermal sensors is an essential tool for evaluating the thermal behavior and sustaining the reliable operation in high-performance and high-power systems. With current technology scaling and integration trends timely and accurate detection of localized heating will be evermore important. In this work, we address the creation of a resource efficient sensor infrastructure for computing systems that are of regular nature, such as logic array-based computing platforms. We propose algorithms to embed thermal sensors into a regular structure to minimize the number of sensors and determine sensor locations required to maintain a given accuracy in temperature sensing for a given design. Our algorithms are tailored for minimal usage of thermal sensors to suit a variety of architectural conditions. For programmable logic arrays the highly application-specific usage of the hardware resources leads to unpredictable thermal profiles. As a result, post-manufacture instantiation of thermal sensors is desired, which in turn demands the use of native hardware resources, which can be scarce. We demonstrate that using our techniques the number of sensors required to monitor a set of hotspots is reduced by 75% on an average, across different sizes of logic arrays for different hotspot distributions compared to a uniform distribution of sensors throughout the fabrics.

Categories and Subject Descriptors: J.6 [**Computer Applications**]: Computer-aided Engineering (CAD):

General Terms: Algorithms, Measurement, Experimentation.

Keywords: Temperature, Sensor, Allocation, Placement.

1. INTRODUCTION

Continuous technology scaling has enabled high logic densities on integrated circuits. FPGA devices [1], [2] are among the first to use the 90nm process. Xilinx and IBM have a roadmap to produce chips at 65 nm. Increase in logic density leads to higher power density, which results in high die temperature. For commercial grade FPGAs the maximum die temperature without performance degradation is reported as 80°C and the absolute maximum temperature is 125°C [3]. An average design mapped onto Virtex-E with 90% device utilization could lead to a die temperature that is 50°C above ambient temperature. Considering the temperature of the case to be 40°C, the die temperature would be 90°C in still air and with a fan it would drop to 75°C [3]. Preventive measures for convective cooling (such as large heat sinks) or air-cooling (such as fans) may not be feasible to deploy in certain embedded applications due to size constraints. Therefore, the operating conditions can be expected to remain well above 75°C.

High temperature has adverse effects on switching speed of transistors, resistance of interconnects and reliability. Therefore, thermal monitoring is vital in understanding the thermal behavior in a system and providing directives for run-time preventive measures such as clock throttling. Successful realization of the latter has been demonstrated for microprocessors [4]. Limited versions of dynamic thermal management can be done for FPGAs using their embedded diode to monitor the temperature. In the Virtex family external pins connect to the temperature-sensing diode creating a remote sensor. The interrupt from the sensor can be used to turn off the clock or turn on a fan [5].

Thermal monitoring for FPGAs is relevant for two reasons: (a) FPGAs dissipate high power and their operating temperature can exceed the critical die temperature in the absence of elaborate cooling mechanisms. The high power dissipation trend and consequent thermal stress is going to become more severe for technology nodes at 65nm and below. (b) FPGAs are often used for rapid prototyping and emulation. Gathering thermal data from the logic array is important to characterize the mapped application. In this paper, we address the problem of allocating and placing thermal sensors for FPGAs, where the following architecture-specific constraints exist:

- Reasonably accurate profiling information about the expected thermal gradients on the system can only be obtained after the system has been manufactured and the particular application, which will utilize the device, is known. Therefore, it is not practical to embed sensors into the system at manufacture time.
- If the thermal sensors will be inserted at the post-manufacture stage they can only be generated using the native resources of the system, i.e. configurable logic blocks.
- Collection and interpretation of data supplied by the thermal sensors will be interleaved with computation.

The exact utilization of programmable components and resulting physical factors such as power and temperature are not known a priori, since the same FPGA device can be programmed to perform various tasks. The Configurable Logic Blocks (CLBs) can be occupied or unoccupied depending on the particular application. The power distribution and power density observed on the same FPGA device can be different for two different applications. As a result, locations of hotspots are application dependent. Moreover modern FPGAs contain embedded DSP blocks and microprocessor cores. These embedded cores can either remain unused or exhibit localized heating if they are utilized by the application. There can be multiple hotspots in the FPGA due to the uneven activity at different parts of the homogenous logic array and stripes of embedded cores. It is difficult to determine locations for thermal

[¶]This work was done while the author was at Northwestern University.

ICCAD'06, November 5-9, 2006, San Jose, CA

sensing at the time of device fabrication. Also in presence of multiple hotspots, a fixed single thermal sensor is a poor representation of the die temperature. Moreover, since the thermal characteristics of the FPGA will greatly depend on the application, thermal monitoring has to be done using native resources only. Hence, placing thermal sensors during manufacturing of the reconfigurable device will not be efficient.

Programmability presents a unique opportunity for embedding thermal sensors into an application immediately before mapping it onto the target device. Once the application to be mapped is determined, the candidate hotspot locations can be identified and sensors can be instantiated using unoccupied reconfigurable logic on the FPGA device [6]. Ring oscillators are used for thermal monitoring and can be implemented on all FPGA architectures [7]. One alternative is to embed thermal diodes on the device. As we have argued, it will not be possible to determine the best locations to insert thermal diodes during manufacturing since the same device can be used for widely different applications. Moreover, ring oscillators are better than thermal diodes because of their fully digital output and more linear characteristics [6, 7]. However, static placement of ring oscillator-based sensors requires significant amount of resources and incurs communication overhead to read back values from sensors. One technique to circumvent the resource overhead is to perform dynamic reconfiguration to insert sensors, take readings, and then vacate the resources occupied by the sensors for the use of the application [7]. However, for a large number of sensors distributed across the device this will incur significant overhead, especially if thermal monitoring is used for taking preventive measures during the run-time of the application. Therefore, it is important to reduce the number of sensors and control their locations (while maintaining a given level of accuracy) in order to reduce the run-time reconfiguration and read back delay in such a scenario. Using less real estate to implement sensors may allow static placement. This also simplifies the microcontroller and peripheral design used to control such statically placed sensors. Hence, there is a need for a systematic approach to determine the minimal number and best locations of thermal sensors in such resource constrained devices.

Given a design exhibiting a certain thermal profile, we propose an algorithm to minimize the number of sensors and determine their placement to monitor the hotspots within a given accuracy. Note that within the same application, number and location of hotspots may shift depending on the input behavior. In that case, multiple instances of the same application and their respective thermal maps would be superimposed to obtain the map of hotspots. We present an efficient *Recursive Bisection* algorithm to create clusters of hotspots. An initial rectangle bounding all hotspots is then recursively bisected into smaller rectangles until their perimeter bound hotspots can be covered by a sensor.

Our specific contributions in this paper are as follows:

- Propose a novel methodology for thermal monitoring for fine grain reconfigurable fabrics,
- Formulate the problem of minimization of sensors required to monitor a given distribution of hotspots within a pre-defined error margin,
- Develop Recursive Bisection algorithm to effectively minimize the number of sensors and determine their placement.

The remainder of the paper is organized as follows: Section 2 gives an overview of the related work. In Section 3.2 we elaborate the paradigm for sensor placement in reconfigurable logic. We discuss our sensor placement algorithm in Section 3.2.4. Section 4 presents our results. We conclude with a summary in Section 5.

2. RELATED WORK

Veluswamy et al. [8] validated the thermal simulator HotSpot for FPGAs against readings taken with statically configured ring oscillators. Recently sensor placement for FPGAs have been proposed in [9, 10].

Our sensor placement technique requires some form of partitioning of the hotspots, which we achieve through our Recursive Bisection method. There are various techniques to generate clusters from a set of elements, such as the K-means clustering algorithm commonly used in data mining [11]. Clustering in the general sense mostly deals with abstract distance functions, which do not directly correspond to actual spatial distribution on the 2-D plane (or 3-D space). In the context of wireless sensor networks, *coverage* refers to the quality of surveillance [12]. In contrast our algorithm aims to determine both the number and the positions to deploy sensors to achieve a successful coverage. The Art Gallery Problem [13] is yet another type of coverage problem often encountered in sensor networks settings. Placing sensors for thermal monitoring is constrained by the range of the sensor in contrast to the "line of sight" of the observer in Art Gallery Problem.

Partitioning by recursive bisection is widely applied in placement algorithms in physical design [14]. During recursive bisection for placement the objective is to minimize the net cut cost and elements can be moved across bisections in each iteration. In our problem, the elements of clusters i.e. hotspots have fixed locations. Using our Recursive Bisection technique we group the elements such that the enclosing rectangle around the hotspots satisfies a given dimensional constraint. The center of the cluster is the location of the sensor. The goodness of the solution is measured in terms of the number of sensors, which is equivalent to reducing the number of clusters obeying the dimensional constraint.

3. THERMAL SENSOR PLACEMENT

In this section we discuss our minimal sensor placement paradigm.

3.1 Thermal Sensors on Reconfigurable Fabric

A widely used technique to implement thermal sensors on FPGAs (post-fabrication) is based on ring oscillators. Transistor switching speed is related to temperature. This property is used to measure temperature by sensing the output frequency of the ring oscillator. The sensor consists of an odd number of inverters connected in a chain and can be implemented in FPGAs using LUTs [8]. The buffered output of the oscillator is used to clock the capture counter, which specifies the oscillation frequency. The thermal sensors can be statically placed or dynamically inserted, operated and eliminated from the circuit using full or run-time reconfiguration using the configuration port capabilities in Xilinx technology [7].

3.2 Minimal Sensor Placement

Our algorithm minimizes the number of sensors and determines their placement to monitor a set of hotspots, where a range of sensitivity for the sensor technology is defined to achieve certain accuracy.

3.2.1 Assumptions and Definitions

We assume that the ring oscillator based thermal transducers will be used as sensors as described in Section 3.1. We refer to placing a regular array of such sensors within a fixed grid across the entire CLB array as the *Grid-based* placement. This is equivalent to the

logic array being partitioned into grids equal to the number of sensors and each sensor being placed at the center of the grid. The Grid-based placement is shown in Figure 1(a). For example Beudo et al. [15] used a 4×8 array of sensors to uniformly cover the XCV800HQ240-4C Virtex FPGA. The CLB array size for this device is 56×84. This corresponds to each sensor being placed in the center of a 14×11 CLB array. We refer to this as a single sensor *covering* a 14×11 CLB array. The temperature read by a sensor placed at the center of the *coverage area* is representative of the temperature of the entire coverage area. However, different points within the coverage area can be at different temperatures. The error in approximating the temperature of the coverage area by the temperature at its center is tolerable if it is below a given error margin ΔT. The largest distance from the sensor at which the error in approximating the temperature by that of sensor is less than ΔT determines the *range* S_R of the sensor. We define *coverage area dimension* C_D as a function of *range* S_R such that a rectangular area with {height, width} $\leq C_D$ centered on a sensor *covers* all hotspots within that area. Thus *coverage area dimension* $C_D = \sqrt{2}.S_R$. Although this type of grid based sensor placement works reasonably well, the number of sensors required increases proportional to the CLB array size. To mitigate the problem of increasing sensor array size, we propose an effective approach, which will be described in the following.

3.2.2 Overview

The designer can perform thermal simulation on a set of applications and representative input data sets to obtain the map of potential hotspots. To effectively monitor this map with minimal resource overhead judicious selection of the number and position of sensors is necessary. Our algorithm solves this sensor placement problem by creating bounding rectangles around hotspots by Recursive Bisection. The sensors are placed at the center of such bounding rectangles. The sensor placement for a given set of hotspots is illustrated in Figure 1(b).

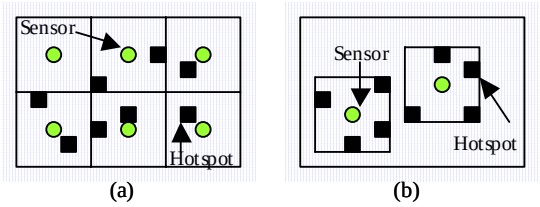

Figure 1. (a) Grid-based placement of sensors. (b) Minimal sensor placement for a set of hotspots. The rectangular coverage area of each sensor is covering four hotspots.

Table 1 shows the number of sensors required to monitor different logic arrays using Grid-based placement.

Table 1. Number of sensors required for different array sizes using Grid-based placement vs. minimized number of sensors for Recursive Bisection (RB) covering up to 50 hotspots on these arrays.

CLB Array	64×42	96×64	128×86	160×110	192×128
Sensors (Grid-Based)	18	40	72	114	160
Sensors - RB for 50 Hotspots	8	19	20	24	30

Using 2 slices per CLB a sensor takes 16 CLBs to implement and the sensor array occupies over 10% of the CLBs. Dynamically reconfiguring a large sensor array can be expensive in terms of reconfiguration time and read back from the sensor counter. We also show the number of sensors required to monitor up to 50 randomly distributed hotspots in the same logic arrays in Table 1. Using our Recursive Bisection algorithm the number of sensors required for 128×86 and 192×128 CLB arrays is 20 and 30 respectively as compared to placing a fixed 72 and 160 sensors using the Grid-based approach.

3.2.3 Problem Formulation

Given,

- a p×q array of configurable logic blocks
- a set of hotspots H = {h_1 (x_{h1}, y_{h1}), h_2 (x_{h2}, y_{h2}), …, h_k (x_{hk}, y_{hk})}, where (x_{hi}, y_{hi}) are coordinates associated with each hotspot
- a sensor with associated maximum coverage area dimension C_D

Our goal is to determine the minimum cardinality set of sensors S = {s_1, s_2, …, s_n} and the position (x_{si}, y_{si}) of each sensor $s_i \in S$ such that:

- each hotspot h_i is covered by some sensor s_j, denoted by the one-to-many relation $s_j \rightarrow H_{sj}\{h_u, \ldots, h_v\}$, s.t. the minimum enclosing rectangle for each H_{sj} is at most of dimension C_D on each side.

3.2.4 Sensor Placement by Recursive Bisection

Our goal is to minimize the number of sensors and determine their placement to cover a set of hotspots. Each sensor has a coverage area dimension C_D. Sensor s_i can monitor hotspots that lie within the coverage area of height and width at most equal to C_D. Our algorithm tries to create a set of rectangles each having their height and width at most C_D containing a set of hotspots. The sensor is placed at the center of such a rectangle. We will show the actual region formed can be smaller than this dimension depending on the location of the hotspots that the sensor covers. That is why we refer to the shape of the region as a rectangle as opposed to a square of maximum dimensions of $C_D \times C_D$. Let us assume that a bounding rectangle R is created around five hotspots h_1, h_2, … h_5. The sensor S is at the center of R. Let the distance of the hotspot h_1 from the sensor be r_1. Similarly the distances of other hotspots are denoted by r_2, … r_5. Since the sides of the bounding rectangle are at most C_D, the maximum distance of any hotspot within R can be S_R from S, if $C_D = \sqrt{2}.S_R$. This is shown Figure 2.

The hotspots h_1 and h_5 are at the two corners and farthest from sensor S. $r_1 = r_5 = S_R$ in this case. The other hotspots are closer to S i.e. $r_{2,\ 3,\ 4} < S_R$. Thus placing the sensor at the center of the bounding rectangle of height and width of at most C_D guarantees that all hotspots within this rectangle are within the range of the sensor.

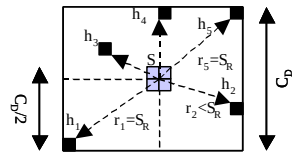

Figure 2. Sensor at the center of coverage area of dimension $C_D \times C_D$.

The algorithm works by recursively creating a bounding rectangle R_i around hotspots and bisecting the rectangle along an edge until both height and width of R_i is less than or equal to C_D. The number of sensors *n* corresponds to the number of rectangles R_p, …, R_q. Each R_i has a list of hotspots L_H^i that its perimeter covers. Two pairs of (x, y) coordinates can be used to represent R_i, which we denote by {(x_{left}^i, y_{bottom}^i), (x_{right}^i, y_{top}^i)}. The steps of the algorithm are illustrated in Figure 3(a) - (f).

First, our algorithm creates the smallest bounding rectangle R_1 around all hotspots. R_1 is then pushed back into a list, which we denote by L_{Rec}. If the height and/or width of R_{Init} is greater than C_D, it is deleted from the list and is bisected along the edge determined by the *Edge Selection* procedure and the point of bisection is determined by the *Bisection Point Selection*. We will discuss the details of the selection mechanisms in the following sub-sections.

After vertical bisection we would get two new rectangles $R_2\{(x_{left}^1, y_{bottom}^1), (x_{bisect}, y_{top}^1)\}$ and $R_3\{(x_{bisect}, y_{bottom}^1), (x_{right}^1, y_{top}^1)\}$. Similarly after horizontal bisection the resulting rectangles are $R_2\{(x_{left}^1, y_{bottom}^1), (x_{right}^1, y_{bisect})\}$ and $R_3\{(x_{left}^1, y_{bisect}), (x_{right}^1, y_{top}^1)\}$. The list of hotspots contained in R_2 and R_3 are updated. This is shown in Figure 3(b). Then, the new rectangles R_2 and R_3 are adjusted such that they form *tight* bounding rectangles around the contained hotspots (shown in Figure 3(c)). Now in the second stage of the algorithm, L_{Rec} has two bounding boxes. The next rectangle of the list is then examined if its sides are $\leq C_D$. If not, the rectangle is bisected, erased from the list and the bisected rectangles are pushed back into the list. If the height and width of the rectangle is within the coverage area dimension C_D the current rectangle is unchanged and the next rectangle is checked. The algorithm continues until it reaches the end of the list at which point we have a set of rectangles which all satisfy the height and the width constraint ($\leq C_D$). These steps are shown in Figure 3(d) - (f). As shown in Figure 3(f) the final list of rectangles are R_4, R_5, R_8, R_9, R_{10}, and R_{11}. Let us represent the final list as L_{Rec}. There are six rectangles and hence, six sensors are necessary to cover all the hotspots. The centers of rectangles R_i' represent the positions of the sensors s_i'. The pseudo code of the algorithm is shown in Figure 4.

During bisection two decisions can be made (i) select the edge to bisect along and (ii) decide the point through which the selected edge is bisected. We have developed techniques to make intelligent choices for both. The different edge and bisection point selection strategies are described in the next sections.

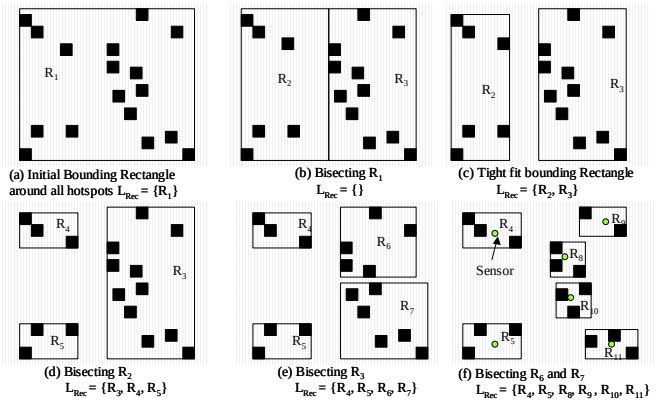

Figure 3. Stages of the Recursive Bisection algorithm.

3.2.5 Edge Selection for Bisection

At each stage of the algorithm a rectangle is bisected if its height and/or width are $>C_D$. If both the height and the width are $>C_D$, the edge to bisect first can be selected in two ways.

Longest Edge: The longest edge of the rectangle is selected.

Look-Ahead: During bisection it is better to have the majority of the hotspots concentrated in one of the bisected rectangles rather than sparsely distributing hotspots in both bisections. Such a sparse distribution may lead to a higher number of sensors each covering fewer hotspots. The Look-ahead edge selection considers bisections in both vertical and horizontal directions. It identifies the number of hotspots that are left on either side of the bisection lines in both directions. The bisection direction, which leads to the largest majority of the hotspots to fall on the same side of the bisection is selected. During this comparison of which direction to pick for bisection, Look-ahead edge selection needs to use an anchor point to perform the cuts in either direction. The particular anchor point (which we refer to as the Bisection Point) will be determined based on the strategy chosen for the Bisection Point selection procedure. We will describe this in detail in the next section.

Minimal Sensor Placement Algorithm

Input: Dimension (Rows, Cols) of the CLB Array
Set of hotspots H and positions of hotspots $h_i(x_i, y_i)$, $h_i \in H$
Coverage area dimension C_D

Parameters: EdgeSelectType, BisectType

1. Create initial bounding rectangle R_{Init} for all the hotspots
2. Create a list of bounding rectangles L_{Rec}
3. Push back R_{Init} onto L_{Rec}
4. List iterator $L_{iterator} = L_{Rec}$.begin()
5. **While** ($L_{iterator}$!= L_{Rec}.end())
   a. Get R_i pointed by $L_{iterator}$
   b. If (Checksides(R_i) = true) { Increment $L_{iterator}$; Continue;}
   c. Bisect(R_i, BisectType, EdgeSelectType)
   d. Assign hotspots of R_i to R_{i1} and R_{i2} to L_H^{i1} and L_H^{i2} respectively
   e. Create tight rectangles R_{i1}' (R_{i2}') containing L_H^{i1} (L_H^{i2})
   f. Push back R_{i1}' and R_{i2}' into L_{Rec}
   g. Delete R_i from L_{Rec} and Increment $L_{iterator}$
   End while
6. Size of the set of sensors S n = Size of L_{Rec}
7. Position of s_i = center of r_i, where $s_i \in S$, $r_i \in L_{Rec}$, $\forall i \in \{1 \ldots n\}$

Output: Number and Position of a set of sensors which covers H

Figure 4. Sensor placement by Recursive Bisection. Depending on the EdgeSelectType parameter either Longest-edge or Look-ahead is selected. Depending on the BisectType either Balanced, Unbalanced or Neighbor-detect method is selected.

3.2.6 Bisection Point Selection

We present three procedures for selecting the bisection point.

Balanced: The center point of the edge is selected.

Unbalanced: The edge of length l is bisected in ratio C_D:($l - C_D$). This results in one of the rectangles (say R_l) having an edge equal to C_D. It ensures that R_l will not be bisected along the current edge anymore and all rectangles are bisected a minimum number of times. This is shown in Figure 5(c).

However, unbalanced bisection makes early decisions as to the size of the rectangle affecting the final solution quality. An example is shown in Figure 6. The cluster of hotspots that can be covered by a single sensor is initially bisected during unbalanced bisection. This results in 3 sensors for balanced vs. 4 sensors for unbalanced bisection. To improve quality, unbalanced bisection is only used at later stages of the algorithm. Unbalanced bisection along an edge is allowed only when the other edge is $\leq C_D$.

Neighbor Detection: The quality of bisection for three different points is evaluated before the actual bisection. An edge can be represented by a pair of {start, end} points. Three different possible bisection points are considered {B_{Left} = (start + C_D), B_{Center} = (start + end) /2, B_{Right} = (end – C_D)} out of which one is finally selected. For each bisection point, there will be two resulting rectangles {R_{l1}, R_{l2}} and their respective hotspot lists {L_H^{l1}, L_H^{l2}}. Two hotspots in a list are *neighbors* if the Euclidean distance between them is less than or equal to C_D and can possibly lie in the coverage area of a single sensor. For each hotspot list L_H^{l1} and L_H^{l2} we find the neighbor count and select the bisection point for which their sum of neighbor count is maximum. Hotspots that are neighbors have a higher probability of being covered by the same sensor. We refer to this strategy as bisection point selection by neighbor detection.

For n hotspots, longest edge and look-ahead edge selection take constant and $O(n)$ time respectively. Balanced and Unbalanced

bisections take constant time and the Neighbor-detect bisection method takes $O(n^2)$ time. The running time for the steps 5d and 5e (Figure 4) is $O(n)$. The steps in the while loop 5a - 5g take $O(n +$ (Edge select, Bisect)) time. In the worst case the while loop can run *n* times. For example the running time for Longest-edge Neighbor-detect sensor placement has a running time of $O(n^3)$ vs. $O(n^2)$ for Look-ahead Unbalanced.

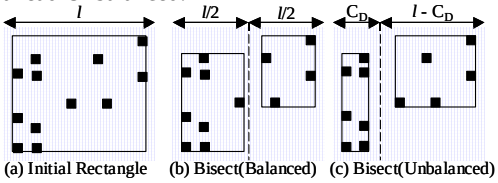

Figure 5. Two bisection alternatives.

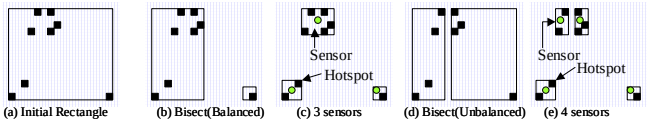

Figure 6. Illustrating problems with unbalanced bisection (a) Initial rectangle (b) Balanced bisection (c) Resulting 3 sensors (d) Unbalanced bisection (d) Resulting 4 sensors.

4. EXPERIMENTAL RESULTS

In the following sections we first discuss our experimental setup and then our results.

4.1 Experimental Setup

First, thermal simulation is performed on the set of applications to be mapped onto the logic array. Effective thermal simulators have been recently proposed [16, 17]. A set of hotspots to be monitored is identified.

Our approach is common in a profile driven paradigm. Similarly, power estimation (e.g. Power Model for VPR) based on representative input patterns is performed. An input should be sustained for several 100us to create a hotspot. With FPGA frequencies in the order of 100's of MHz, if there are input patterns with such duration not captured, this means the designer missed a typical input behavior. We assume this should not happen. If a hotspot falls outside all coverage areas, the nearest sensor will still pick up the temperature trend of the logic, however error margin will degrade.

The coverage area of the thermal sensor is then estimated. In previous work where regular Grid-based placement was proposed [15], a sensor is placed in the center of a 14×11 CLB array. In order to be able to make a comparison, we have derived the required coverage area dimension of a sensor from this placement. We set the largest CLB array size that the sensor can cover as a 12×12 square. As a result we define the coverage area dimension of the sensor, C_D as 12. The sensor is placed at the center of such a 12×12 CLB array. Using our Recursive Bisection algorithm the number and locations of the thermal sensors are determined. The sensors can then be placed using run-time reconfiguration or embedded statically if the CLB utilization of the design has enough slack. The next section presents our experimental results.

4.2 Results

We generated two types of inputs to evaluate the effectiveness of our algorithm. Part of the input maps of hotspots were created by randomly assigning 5 to 100 hotspots on a given logic array. We also generated thermal profiles for MCNC benchmarks [18]. The netlist of CLBs was placed and routed using VPR [19]. Power Model [20] was used to calculate the power of each CLB and net, based on the switching activities at the nodes. The data from Power Model was used to perform thermal simulation using the thermal simulator HotSpot [16]. The hotspot distributions observed in those simulations were then used as input to our algorithm.

For the random hotspot maps we used the following methodology. For each array size, we created hotspot maps containing 5 to 100 hotspots in increments of 5. Note that for smaller array sizes the upper bound on the number of hotspots was less (e.g. 50 for the smallest array). Each of these configurations was repeated 10 times using different *seeds*, e.g. 5 hotspots were randomly distributed on an array in 10 different ways and similarly for all other configurations. The rationale behind this setup is to evaluate the sensitivity of our algorithm to a wide variation of inputs and design characteristics. These experiments help us evaluate a large spectrum of cases from large arrays with very few hotspots (such as 192X128 and 5 hotspots) to small arrays with a high number of hotspots (such as 64X42 and 50 hotspots) and many possibilities inbetween. The range for the random choice of number of hotspots vs. the array size is shown in Table 2. We have experimented with logic array sizes of 64×42, 96×64, 128×86, 160×110, and 192×128 CLBs approximating the dimensions of various chips from the Xilinx Virtex-4 family.

While the input set described above was useful for assessing the robustness of our algorithm, we also experimented with thermal data derived from actual power and placement characteristics of real benchmarks. We have added hotspot maps from actual thermal simulations of the MCNC benchmark suite as described earlier.

Table 2. Logic array sizes vs. number of hotspots assigned.

CLB Array	64×42	96×64	128×86	160×110	192×128
Hotspots	5-50	5-75	5-75	5-100	5-100

We compare the number of sensors obtained by using different Edge Selection (longest edge, and look-ahead) and Bisection Point Selection Techniques (balanced, unbalanced, and neighbor detect) for different number of hotspot assignments for each logic array. Table 3 shows the results for the randomly generated hotspot distributions. Due to space constraints, out of 6 possible variations to our algorithm, we present the number of sensors for look-ahead balanced (*lk-bal*), look-ahead unbalanced (*lk-unbal*) and longest edge neighbor detect (*lg-nb-det*). These combinations form the most interesting set of results. For example for 192×128 logic array with 50 and 100 hotspots randomly distributed, the number of required sensors using {*lk-bal*, *lk-unbal*, *lk-ng-det*} are {34, 34, 30} and {54, 52, 44} respectively. It can be seen that *lk-unbal* is better than *lk-bal*. This is because unbalanced creates less number of rectangles (so less number of sensors) than balanced bisection. *lg-nb-det* method gives the least number of sensors when compared to *lk-bal* and *lk-unbal*. Creating bounding rectangles such that they preserve the highest neighbor count indeed creates a better solution. Based on these results, we observed that longest edge neighbor detect strategy performs best.

Next we present our results incorporating the hotspot distributions obtained from thermal simulation of MCNC benchmarks placed and routed on representative CLB arrays. We obtained the number of sensors required to monitor these benchmarks on each of these arrays. Figure 7 illustrates the overall average number of sensors required for monitoring these benchmarks as well as the hotspot distributions presented in Table 3. Figure 7 presents results for *lk-bal*, *lk-unbal*, and *lk-nb-det*. The average number of sensors required across array sizes {64X42, 96X64, 128X86, 160X110,

and 192X128} using *lg-nb-det* are {7.54, 15.75, 17.48, 26.01, and 28.78} respectively. Longest edge neighbor detect (*lg-nb-det*) performs the best, which is consistent with our previous observations. We conclude that longest edge neighbor detect proved its robustness during our extensive evaluation using synthetic inputs. It also demonstrated its effectiveness when applied to realistic benchmarks. Therefore setting the parameters EdgeSelectType = Neighbor Detection, and BisectType = Longest Edge, in our algorithm (shown in Figure 4), gives the best overall performance. There is a tradeoff between complexity and quality of the results. Both *lk-bal* and *lk-unbal* have a running time of $O(n^2)$, whereas that of *lg-nb-det* is $O(n^3)$. On the other hand, the quality of the result in the case of *lg-nb-det* is superior compared to the other two requiring 16.6% less sensors compared to *lk-bal* on average, and 14.6% fewer sensors compared to *lk-unbal* on average.

Table 3. Number of sensors obtained using Recursive Bisection for different number of hotspots assigned different logic array sizes.

CLB	96X64			128X86			160X110			192X128		
Hotspots	lk-bal	lk-unbal	lg-nb-det	lk-bal	lk-unbal	lg-nb-det	lk-bal	lk-unbal	lg-nb-det	lk-bal	lk-unbal	lg-nb-det
5	5	5	5	5	5	5	4	4	4	5	5	5
10	7	7	7	9	9	8	9	9	9	8	8	8
15	9	9	9	11	11	10	13	13	13	11	11	11
20	10	10	10	12	12	11	16	16	14	17	17	16
25	13	12	11	15	15	13	18	18	16	19	19	18
30	15	14	13	17	17	13	23	22	20	23	23	22
35	15	15	14	20	20	15	23	23	21	25	25	23
40	19	19	16	22	22	19	25	24	21	28	28	27
45	19	19	17	23	20	19	28	27	23	31	31	30
50	22	21	19	26	25	20	31	29	24	34	34	30
55	23	22	21	31	29	20	35	33	27	39	37	30
60	26	24	21	32	29	25	36	34	31	40	39	31
65	27	25	21	32	29	26	36	35	34	42	41	30
70	25	22	24	33	29	28	39	37	32	44	44	35
75	28	25	23	29	26	29	41	40	34	47	46	37
80	Array Size Not Applicable			Array Size Not Applicable			45	42	36	48	46	37
85							48	46	39	52	49	39
90							51	50	39	53	51	44
95							51	50	40	51	49	43
100							53	51	44	54	52	44
Avg	**17.5**	**16.6**	**15.4**	**21.1**	**19.9**	**17.4**	**31.3**	**30.2**	**26.1**	**33.6**	**32.8**	**28.0**

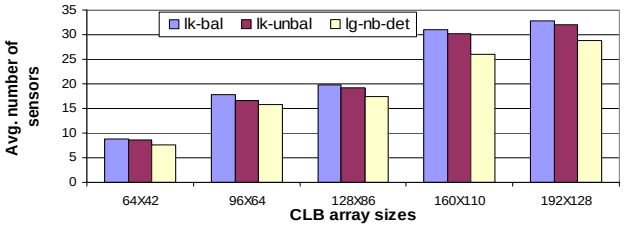

Figure 7. Average number of sensors for a range of hotspot distributions in different logic arrays.

Further, the average number of sensors across all logic arrays and hotspot distributions using our best technique is found to be 19.11. In comparison, for Grid-based placement the average number of sensors required across all logic arrays is 80.8 (see Table 1). This translates to a 75% saving on an average in the number of required sensors using our methodology.

5. CONCLUSIONS

In this paper we have proposed a novel methodology to minimize the number of sensors and determine their location for thermal monitoring of hotspots. We have formulated the sensor allocation and placement problem and presented an efficient Recursive Bisection algorithm to achieve this goal. Given a distribution of hotspots, our algorithm is indeed effective in minimizing the number of sensors and determining their location. We have evaluated our algorithm for different edge and bisection point selection techniques and conclude that longest edge neighbor detect (*lg-nb-det*) provides best results when compared to other techniques. Using our methodology we demonstrate that the number of sensors required to monitor a set of hotspots is reduced by 75% on an average, across different logic arrays for different hotspot distributions when compared to a regular distribution of sensor array previously proposed in the literature.

6. ACKNOWLEDGEMENTS

This research is supported in part by the NSF Career Award CNS-0546305 and the Northwestern University Alumnae Association Research Initiation Award.

7. REFERENCES

1. Altera. *Excalibur Device Overview*.
2. Xilinx, *PowerPC in Virtex-4 FX*.
3. Lesea, A. and M. Alexander. *Powering Xilinx FPGAs*. 2002
4. Gunther, S., et al., *Managing the impact of increasing microprocessor power consumption.* Intel Technology Journal, February 2001.
5. Xilinx. *Answers Database: Virtex/Virtex-E/Virtex-II/Virtex Pro/Virtex-4 - What are temperature-sensing diode pins (DXP and DXN, TDN and TDP)?* 2005
6. Lopez-Buedo, S., J. Garrido, and E.I. Boemo, *Thermal Testing on Reconfigurable Computers.* IEEE Design and Test of Computers, 2000. **17**(1): p. 84-91.
7. Lopez-Buedo, S., J. Garrido, and E.I. Boemo, *Dynamically Inserting, Operating, and Eliminating Thermal Sensors of FPGA-based Systems.* IEEE Transactions on Components and Packaging Technologies, 2002. **25**(4): p. 561- 566.
8. Velusamy, S., et al. *Monitoring Temperature in FPGA based SoCs*. in *International Conference on Computer Design*. 2005.
9. Mukherjee, R., S. Mondal, and S.O. Memik. *A Sensor Distribution Algorithm for FPGAs with Minimal Dynamic Reconfiguration Overhead*. in *To appear in International Conference on Engineering of Reconfigurable Systems and Algorithms*. 2006.
10. Mondal, S., R. Mukherjee, and S.O. Memik. *Fine-Grain Thermal Profiling and Sensor Insertion for FPGAs*. in *IEEE International Symposium on Circuits and Systems* 2006.
11. MacQueen, J. *Some Methods for Classification and Analysis of Multivariate Observations*. in *Fifth Berkeley Symposium on Mathematical Statistics and Probability*. 1967.
12. Meguerdichian, S., et al. *Coverage Problems in Wireless Ad-hoc Sensor Networks*. in *INFOCOM* 2001.
13. Chvatal, V., *A Combinatorial Theorem in Plane Geometry*. Journal of Combinatorial Theory 1975. **18**: p. 39-41.
14. Sherwani, N.A., *Algorithms for VLSI Physical Design Automation*. 1995, Norwell, MA: Kluwer Academic Publisher.
15. Lopez-Buedo, S. and E. Boemo. *Making Visible the Thermal Behaviour of Embedded Microprocessors on FPGAs: a Progress Report*. in *International Symposium on Field Programmable Gate Arrays*. 2004.
16. Skadron, K., et al. *Temperature-Aware Microarchitecture*. in *International Symposium on Computer Architecture*. 2003.
17. Yang, Y., et al. *Adaptive Chip-Package Thermal Analysis for Synthesis and Design*. in *Conference on Design, Automation, and Test in Europe*. 2006.
18. Yang, S. *Logic Synthesis and Optimization Benchmarks*. 1991: Microelectronics Center of North Carolina.
19. Betz, V., J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. 1999: Kluwer Academic Publishers.
20. Poon, K.K., *Power Estimation for Field Programmable Gate Arrays*, in *Dept. of Electrical and Computer Engg*. 1999, University of British Columbia.