Hotspot Mitigation using Dynamic Partial Reconfiguration for Improved Performance

Adwait Gupte
Iowa State University
Department of Electrical & Computer Engineering
Ames, IA, USA
Email: adwait@iastate.edu

Phillip Jones
Iowa State University
Department of Electrical & Computer Engineering
Ames, IA, USA
Email: phjones@iastate.edu

***Abstract*—As the chips get denser and faster, heat dissipation is fast turning into a major problem in development of ICs. Non-uniform heating of chips due to hotspots is also an area of concern and much research. In this paper, we propose an adaptive method which takes advantage of the self-reconfiguration capability of modern FPGAs to mitigate hotspots. We adapt the floor plan of the IC in response to the current use and ambient conditions on the fly. It is most applicable to paradigms such as Network on Chip (NoC) that allow separation of communication and computation and allow communication between modules to be abstracted away. We achieve a reduction of up to 8 °C in the maximum temperature of a hotspot using typical power numbers. Alternatively, by increasing the frequency, we achieve a 2-3 times increase in throughput while maintaining the same maximum temperature.**

I. Introduction

Advances in technology continue to make the manufacture of denser chips possible. As chips get smaller and denser, power dissipation becomes more difficult. It is becoming increasingly important to be able to monitor as well as manage the temperature of a device in order to ensure optimum performance as well as longevity [1].

The temperature of an Integrated Circuit (IC) is a function of the switching activity in the chip. The switching activity of each point on the chip may differ. For example, in an IC that encrypts data, the part of the IC that generates a random number once for each message will not get as heated as other parts that XOR each bit of the message with the bits of the key. Hence certain areas of the chip get heated more than others. Such spots of higher temperature in a chip are known as "Hotspots". Studies show that an increase in junction temperature by 30 °Celsius decreases the Mean Time Between Failure (MTBF) by a factor of 10 [2].

In the software domain, works such as [3] by Powell et al have proposed methods that migrate tasks from one processor to another in a Chip MultiProcessor (CMP) system when the processor heats up to a threshold value. In this paper, we propose to apply a similar concept to mitigating hotspots on Field Programmable Gate Arrays (FPGAs). FPGAs are SRAM based reconfigurable hardware devices. Besides being able to be completely reconfigured when required, modern FPGAs also allow partial reconfiguration where only a part of the FPGA is reconfigured while the rest of the FPGA remains active and continues to provide services. Dynamic Partial Self-Reconfiguration refers to the ability of modern FPGAs to themselves control the reconfiguration of a part of the FPGA while the rest of it continues to provide services.

In this paper, we use the dynamic partial reconfiguration capability of an FPGA to handle hotspots as they occur on the IC. The basic idea is to divide the design into smaller, almost equal sized modules. Each module has a built in capability for measuring temperature(such as in [4]). Whenever the temperature of a given region reaches a threshold value, dynamic partial reconfiguration is used to swap that module to a different area. Thus distributing the heat more evenly over the surface of the FPGA. Paradigms such as Network-on-Chip (NoC) [5] allow such a scheme to be implemented by separating communication and computation. The two most important advantages of implementing such a scheme are 1) it decreases the maximum temperature of the hotspots on the chip and 2) it allows a substantial increase in performance while still staying within the temperature budget.

In the subsequent sections we explain our method in more detail.

The remainder of the paper is organized as follows: Section II discusses some closely related works and helps put our work into perspective within the larger body of thermal management work. Section III describes our approach in greater detail. Section IV then discusses and analyzes the results of our experimentation. Section V concludes the paper by reiterating some our primary contributions and pointing to paths of future work.

II. Related Work

In the following section we discuss various past works closely related to this paper. First we discuss the various hardware based techniques that have been used to 1) Detect hotspots and 2) Avoid/mitigate hotspots. Then we discuss some software schemes used in CMP systems that are similar to the one we proposed for reconfigurable hardware. Next we discuss some methods for measuring temperature at various points in the IC. We conclude this section with a comparison of the various scenarios in which the previous methods can and cannot be applied and compare this with the applicability

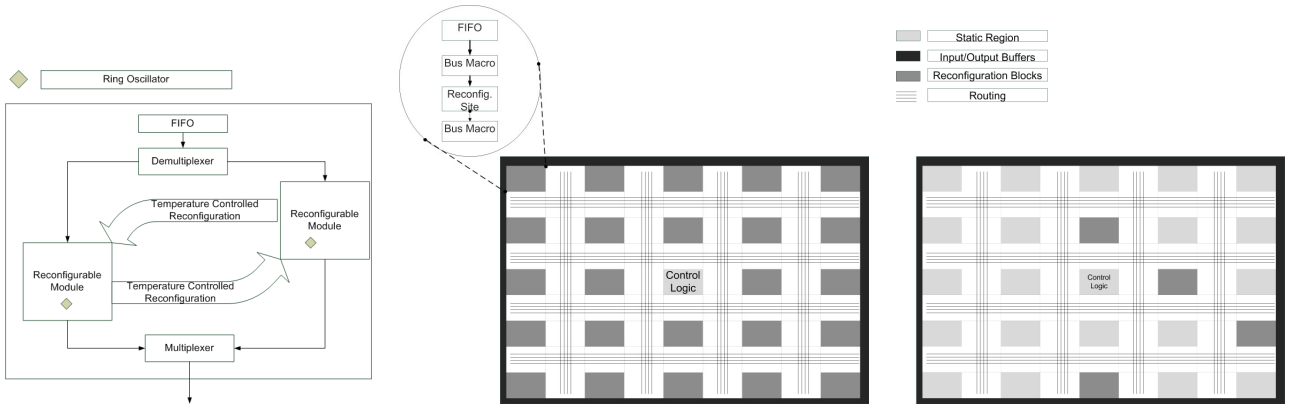


Fig. 1. Figure 1(a) shows a limited implementation of the proposed idea which can be easily implemented with the technology currently available. Figure 1(b) shows a more complete application of the proposed method in which the whole design is divided into modules and each module can be placed anywhere on the chip. Figure 1(c) shows a generalized form of the approach with both static and reconfigurable modules in the design.

of our work.

Hardware Techniques: In [6], Huang et al have described a complete modeling methodology that allows the thermal behavior of a design to be simulated, and hotspots to be identified early in the design process. This allows designers to then take steps which can reduce/avoid these hotspots in the design. In [7], Mukherjee and Memik describe an integrated approach for thermal management that models thermal behavior of the chip at design time. They use this information in all aspects of synthesis to arrive at a thermally optimized implementation that keeps the chip cooler as well as mitigates hotspots. In [8], Hung et al present a genetic algorithm for floor planning that takes thermal factors into account. In [9], Schaffer and Kim propose techniques that are applied at the floor planning stage of a design to avoid hotspots. Their method models and identifies hotspots and then depending upon the timing budget, rearrange the floor plan to keep hotspots far from each other. Also, the hotspots are surrounded with cooler areas of the chip to improve dissipation. Heating in FPGAs is a function of the switching activity. In [10], Siozios and Soudris use estimated switching activity rates to place and route a design so as to evenly distribute the activity and hence achieve a more balanced distribution of heat. Hung et al propose a novel method of taking into account the communication between various processing elements in a Network on Chip (NoC) to control the creation of hotspots in an IC [11]. Link et al propose a runtime reconfiguration based approach for mitigating hotspots in [12]. The primary differences between their work and ours are: 1) We adaptively reconfigure using a temperature threshold trigger while they do so using a periodic trigger, 2) They reconfigure the entire FPGA while we only swap two modules at a time thus decreasing the time spent in reconfiguration, 3) They evaluate translations and rotations on the floor plan, while we swap the modules in a greedy manner with the coolest module 4) This work delves deeper into the details of the possible performance gains and overheads associated with such an approach.

Software Techniques: Powell et al were the first to suggest a method they called "Heat and Run" in which tasks that heated specific areas of a processor were migrated to different processors after the first processor reached a threshold temperature [3]. Thus, the tasks moved around the CMP from one processor to another, evenly distributing the heat across the IC. [13] is another work that similarly suggests migration of tasks to different processors in a multi processor system to avoid overheating of a single processor. We propose to use a similar method in dealing with hotspots in hardware by using the runtime reconfigurability currently available on some modern FPGAs.

Measuring Temperature: Since we trigger reconfigurations using a thermal threshold, it is critical for this method that the temperature of various parts of the IC be measured. Velusamy proposes one method of accurately measuring the temperature of various parts of an FPGA in [4]. Similarly Lopez-Buedo suggests a ring oscillator based method for measuring temperatures on an FPGA in [14]. The ring oscillator method can be used to trigger the reconfiguration in our work.

For many ICs, the exact hotspot may vary according to the manner in which the chip is being utilized and/or have many hotspots. In such cases, applying optimization at the design stage for the worst case scenario may not yield the best results. For these scenarios our method could be beneficial since it changes the floor plan of the IC on the fly, thus adapting to the changing conditions on the chip. The next section explains our method in more detail.

III. Partial Reconfiguration for Mitigating Hotspots

In this section we provide a detailed description of our method for mitigating hotspots. We first discuss the motivation for this work in light of many previous works concentrating on this problem. We then describe a spectrum of levels at which our method can be applied, from just two reconfigurable sites to being able to move any module anywhere in an NoC type

paradigm. Finally we discuss some concerns related to our approach and their possible solutions.

A. Motivating Examples & Benefits

The temperature of an IC depends on the function of the chip as well as the data it is operating upon. Taking the simple data encryption example from Section I forward, if the bits of the message are the same as the random key, the output of the XOR will always be zero. However, if the message bits are such that they cause the XOR output to flip each clock cycle the switching activity is significantly higher and hence the IC will get hotter. Similarly, in multi mode chips, which areas of the chip get heated depends on the way the chip is being used. For example, if an IC is capable of decompressing JPEG as well as PNG images, depending on which image type needs to be decompressed, different areas of the chip will get heated up. Thus the occurrence of a hotspot is not a certainty but a matter of probability dependent upon some factors that are out of the designers control.

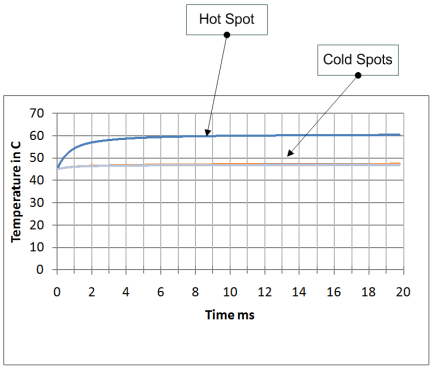
In both these cases, techniques which statically identify hotspots and try to avoid them with floor planning might not be effective. In the first case, floor planning to keep the XOR area away from other hotspots may cause unnecessarily long paths since data which causes such rapid switching may not occur very frequently. In the second case, floor planning will try to keep the JPEG and PNG regions as far away from each other as possible, assuming both will get heated up. This may prevent certain common modules between them from being shared because of longer routing paths. On the other hand, our method dynamically adapts to the prevalent conditions to avoid hotspots.

B. Approach

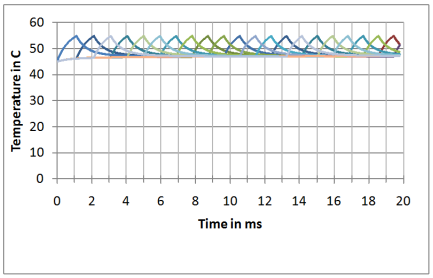
The basic idea we propose in this paper is to swap hot and cool modules around the chip based on thermal triggers in order to distribute the heat evenly thus mitigating hotspots. The implementation of this basic idea could on the one extreme be two modules, one "hot" and the other "cool", implemented as swappable while the rest of the design is static. On the other extreme, every module in the design could be swappable. Figure 1 shows these extremes as well as a more generalized implementation that we use in this subsection to explain our approach in detail.

Figure 1(a) shows the first extreme implementation where there are just two modules that have been implemented as reconfigurable modules. A demultiplexer is used at the input of the modules to guide the incoming data to the correct location of the module. When reconfiguration is triggered, there is a brief period of no-service. The FIFO buffers the data arriving during this period. Ring oscillators are placed in the modules to measure temperatures of corresponding parts of the chip so as to trigger reconfiguration when needed.

Abstracting these low level details away, Figure 1(b) shows another extreme of the proposed approach. Here, except for the control logic, all the modules have been implemented as equal sized reconfigurable modules. The demultiplexers



(a) Without Reconfiguration



(b) With Reconfiguration

Fig. 2. Figure 2(a) shows the change in temperature of the hotspot with time without any reconfiguration. Figure 2(b) shows the change in temperature of the various sites with time. As seen here, the temperature of each of the sites periodically rises and falls as the "hot" module is moved about.

are abstracted into communication channels between modules. Any module can be swapped with any other module. Such an implementation provides more flexibility that Figure 1(a), but also comes with a larger overhead.

Figure 1(c) shows a more generalized implementation of our approach where some selected modules are implemented as swappable modules while others are static. This implementation still gives the designer maximum flexibility, by allowing them to pick the exact modules which are amenable to being implemented as reconfigurable regions and the others are implemented as static. Although the figure shows the reconfigurable modules to be the same size, this is not a absolutely essential. As long as there is a pair of "cool" and "hot" modules of the same size, the approach will work. However, larger the number of options for placing a given "hot" module, more effective this approach becomes as shown in Section IV.

Figure 2(b) shows an example of this approach being applied to a circuit. As seen in the figure, whenever the "hot" module reaches the threshold temperature it is reconfigured. This can be seen from the fact that the original position of that module starts cooling while another location starts heating.

C. Desired Application Attributes & Challenges

Meeting Timing: One of the main concerns with swapping modules is the lengthening of communication paths between modules. In designs that are tightly constrained in terms of timing, the tools find a very specific placement of modules

based on their communication with each other. In such designs, if the modules are moved from one place to another in order to mitigate hotspots, the routes between the modules will lengthen and cause the design to fail the timing requirements. Hence the timing requirements on the design should have enough slack to allow modules to be moved around the FPGA while still meeting timing. Some possible solutions for this problem could be 1) Using an NoC paradigm that abstracts away the communication timing requirements and decouples the communication from computation, 2) Placing relative location constraints on time critical modules that limit the sites in which these modules can be placed during module swapping.

Need for Manual Placement: Current tools do not support automated floorplanning of reconfigurable modules. Such modules need to be manually placed in the appropriate spots by the user. Although this can be somewhat tedious, it is not impossible within the current state of the art. Tool support for partial reconfiguration has been steadily improving making this process continuously more practical.

Reconfiguration Overhead: Reconfiguration requires a finite amount of time to take place. As a result, there will be a small period of "no service" for modules that are being swapped. If a module heats quickly, it will be frequently shifted around the IC resulting in having many such periods of "no service". The FIFOs shown in Figure 1 have been added to buffer the inputs to the modules under reconfiguration.

Jitter due to Reconfiguration: Since the reconfiguration is a function of how fast a module is heating the chip, the fraction of time spent in reconfiguration will vary. The outputs from the modules cannot be guaranteed to arrive at a certain time, instead they will arrive anytime within a finite range. So an application needs to be able to tolerate jitter both between modules as well as on the primary outputs of the design.

Modularity: For best results, the design should allow itself to be divided into small, almost equally sized modules. The smaller the size of these modules, smaller is the impact of reconfiguration time on performance, as shown in Section IV. However this is not a requirement. There can be multiple discrete sizes of modules. As long as there are pairs of "cool" and "hot" modules of the same size, the scheme will work. The efficiency of the scheme may however suffer since the number of possible sites for a swappable module reduces.

Multiple Configuration Files: The current structure of configuration files (bitfiles) specifies the area of the FPGA it targets within its header. The proposed method will hence require multiple bitfiles to be maintained for a module, targeting a different location. This problem could perhaps be solved with changes to the configuration structure of the FPGA to allow partial bitfiles to target any location by simply updating the header.

Saving States: Modules which use pipelining or other designs requiring the state to be saved present an additional difficulty when swapping them. One possible solution for this could be to design these modules such that their state can be saved and loaded. The state can then be saved before swapping and restored when reconfiguration is completed.

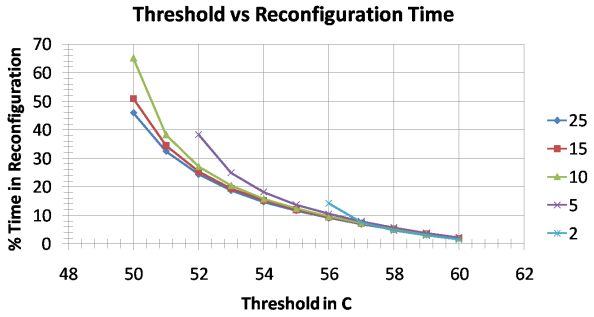


Fig. 3. The graph shows the amount of time spent in reconfiguration (as a % of total time) vs different thresholds. As the threshold is made lower, the time spent in reconfiguration grows exponentially.

IV. Results & Analysis

We implemented Partial Reconfiguration on a Xilinx ML505 board in order to measure the time taken for reconfiguration. Having obtained the time for reconfiguration, we then modified HotSpot 5.0 [6] to simulate swapping of "hot" and "cool" modules. HotSpot takes the power consumption of each module in the design as an input. We modelled the reconfiguration of two modules by exchanging their power consumption in HotSpot. In this section we first present the main simulation parameters used for HotSpot. Next we evaluate two key metrics: 1) reduction in the maximum temperature of a hotspot and 2) increase in throughput by allowing higher frequencies to be used within a thermal budget, as compared to a fixed design. The last subsection analyzes the resource overheads and requirements of our method.

A. Key Simulation Parameters

In [15], Hamann et al claim that a realistic estimate for power density for a hotspot is about 185 W/cm^2 while that for a cool spot is 21 W/cm^2. We used these power settings for a chip divided into 25 modules of equal size. One of the modules was set as a hotspot while all the others were set as cool spots. An ambient temperature of 45°C is used. The size of the modules is half the size of a microblaze (approximately 262 slices) unless otherwise specified.

B. Reduction in Maximum Temperature

M. Pecht references in [1] a rule of thumb that states for every 10°C decrease in the junction temperature, the predicted failure rate reduces by half. This shows an important relation between the temperature of the chip and it's reliability.

Figure 3 shows the relationship between the threshold temperature used to trigger reconfiguration and the fraction of time spent in reconfiguration. The maximum temperature attained by the hotspot without using reconfiguration was 60 °C. The theoretical minimum temperature at which the chip can be maintained is the temperature of the "cool" modules on the chip. Practically, this may not always be possible depending upon the number of swappable modules

available etc. In the simulation experiments, the least threshold that could be maintained was found to be 10°C below the maximum temperature. The fraction of time spent in reconfiguration increases exponentially as the threshold decreases which adversely affects the performance. Thus, even though it might be possible to reduce the maximum temperature by 10 °C in this case, it might not be the most optimal thing to do for the performance

Due to form factor and/or weight constraints, heatsinks may not be desirable for some embedded applications. Using reconfiguration in this manner may bring the maximum temperature of the hotspot to a level where a heat sink is not required depending on the thermal budget.

Different temperatures in different parts of a chip cause the switching speeds of transistors to vary which could cause the timing difficulties. Our method brings the temperature of the hotspot closer to the temperatures of the rest of the chip.

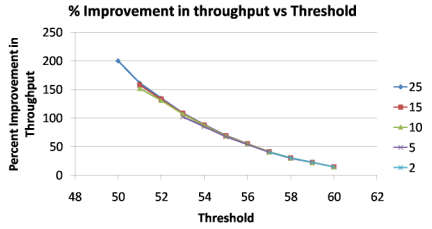
C. Improvement in Throughput

Figure 3 shows the amount of time spent in reconfiguration for a range of temperature thresholds using the parameters mentioned in Section IV-A. It can be seen that the fraction of time spent in reconfiguration can become unreasonably large (up to 65% for 10 °C reduction in hotspot temperatures for 10 modules). However, for a given thermal budget, in the absence of reconfiguration, the only other method to stay within the budget is to reduce the frequency of operation. Hence, the fact that some time is being spent in reconfiguration is offset by the fact that the operating frequency can be increased while staying with the thermal budget. For a given thermal budget (threshold), Figure 4 shows the improvement in throughput obtained by using reconfiguration and a higher frequency of operation versus using a lower frequency in a fixed design. As seen in the figure, for certain configurations, the improvements are negative (ie the throughput degrades rather than improving) but as the number of reconfigurable sites increases, the graph largely remains in the positive for module sizes of a microblaze or less. Thus, for an adequately large number of reconfigurable sites and small enough module size, our method always gives a better throughput than a fixed design that must use a reduced frequency.

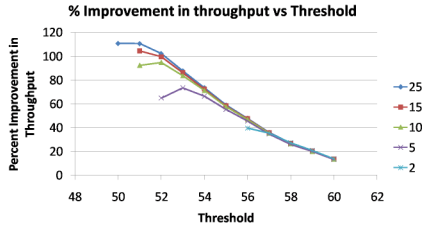
Comparing Figures 4(a), 4(b) and 4(c), it can be seen that as the size of the module increases, the performance of the method becomes worse. Thus, it is important to be able to divide the design into small enough modules. This allows the time spent in reconfiguration to be small and gives larger improvements in throughput as well as reducing the resource overheads (see next subsection).

D. Resource Overhead

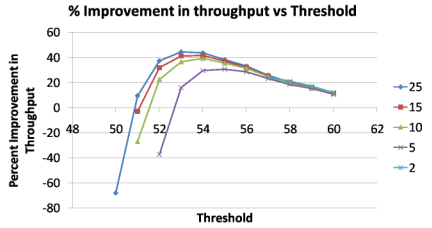
There are brief periods of no-service in the proposed method due to reconfiguration. Hence, it is necessary to buffer any data destined for a module under reconfiguration. The FIFOs needed to buffer this data add memory requirements to the design. In a Network-on-Chip (NoC) type paradigm, routing resources between various modules are already present and



(a) Module size of 1/4 of a microblaze



(b) Module size of 1/2 of a microblaze



(c) Module size of a microblaze

Fig. 4. The increase in throughput with various thresholds taking into account the time spent in reconfiguration for different module sizes.

hence this method adds no resource overhead in routing. On the other hand, if this method is implemented in a non-NoC paradigm, extra routing resources will be needed to redirect data to the appropriate locations after a module has been moved.

Memory Overheads

Consider a module size of 525 slices, which is large enough to fit an entire 32 bit Microblaze Microprocessor. On a Virtex LX50T, the partial bitstream for this would be approximately 133 KB. The ICAP is capable of configuring an FPGA 4 bytes at a time at 100 MHz, thus the total time spent for one reconfiguration would be 0.34 ms. Since two modules are swapped, the total time spent for a swap is 0.68 ms. Assuming an arrival rate of λ for data, the size of the FIFO needs to be 0.68ms * λ.

For example, in a design running at 200 MHz, if data arrives every clock cycle during reconfiguration,($1/\lambda$ is 5 ns), a FIFO size of 132 kilobits is required for a 1 bit signal for 1 module. It's important to note that this is a much worse than average scenario since we're considering the size of the modules to be as large as a microprocessor which will usually not be the case if the design is properly divided into modules.

Rather than over provision the resources by adding FIFOs to every module, a central, shared set of FIFOs can be used. From Figure 2 it can be seen that the thermal time constant

(the amount of time taken for the temperature to rise to 63% of the maximum, from any starting point) is to the order of 20 ms. Thus, we can easily assume that a hotspot will always remain in its location for 1 ms before needing to be moved again. Thus, with one hotspot, it is possible to have only one pair of shared FIFOs for all the modules and still be able to implement this method. As the number of hotspot increases however, more FIFOs will be required.

Consider a scenario where there is one hotspot and the maximum incoming width of any module is 32 bits wide. When a central set of FIFOs is used, 2 FIFOs of size 32 bits would be required to be able to buffer the signals destined for both the modules which are being swapped. This adds up to 8480 kilobits which is can be accommodated in all the larger chips of the Xilinx Virtex 5 family. If the number of hotspots is more than 1, the worse case requirement will grow linearly with the number of hotspots.

Logic Resource Overheads

As seen in figure 1(a), additional logic resources are required to implement the required multiplexers and demultiplexers. The number of inputs/outputs on the multiplexers/demultiplexers is equal to the number of reconfigurable sites. The number of multiplexers/demultiplexers themselves is equal to the total no. of inputs/outputs of all the reconfigurable modules combined. In addition to this, logic resource are also required to support bus macros for all inputs and outputs for reconfigurable region. Extra logic is also needed to control the reconfiguration and to implement the ring oscillators.

V. Conclusion

We have proposed and evaluated a method for mitigating hotspots using the dynamic partial self-reconfiguration capabilities available in some modern FPGAs. Our approach swaps "hot" and "cool" modules at run time using a temperature threshold to trigger reconfiguration, thereby allowing an FPGA floor plan to adapt to its current thermal conditions. Using hotspot to run evaluation experiments, our approach shows a reduction in hotspot temperature by up to 8 °C, and an increase in application throughput by up to a factor of 2-3 times that of a fixed design for a given thermal budget

References

[1] M. Pecht, Ed., *Handbook of Electronic Package Design*. Mercel Dekker, 1991.

[2] N. Semiconductor, "Understanding integrated circuit package power capabilities," in *[Online] Available: www.national.com*, Apr. 2000.

[3] M. D. Powell, M. Gomaa, and T. N. Vijaykumar, "Heat-and-run: leveraging smt and cmp to manage power density through the operating system," in *ASPLOS*, 2004.

[4] S. Velusamy, W. Huang, J. Lach, M. Stan, and K. Skadron, "Monitoring temperature in fpga based socs," in *International Conference on Computer Design*, 2005.

[5] S. Kumar, A. Jantsch1, J.-P. Soininen, M. Forsell, M. Millberg, J. berg1, K. Tiensyrj, and A. Hemani, "A network on chip architecture and design methodology," in *International Symposium on VLSI*, 2002.

[6] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, "Hotspot:a compact thermal modeling methodology for early-stage vlsi design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 5, May 2006.

[7] R. Mukherjee and S. O. Memik, "An integrated approach to thermal management in high-level synthesis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 11, Nov. 2006.

[8] W.-L. Hung, Y. Xie, N. Vijaykrishnan, C. Addo-Quaye, T. Theocharides, and M. J. Irwin, "Thermal-aware floorplanning using genetic algorithms," in *International Symposium on Quality Electronic Design*, 2005.

[9] B. C. Schafer and T. Kim, "Hotspots elimination and temperature flattening in vlsi circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 11, Nov. 2008.

[10] K. Siozios and D. Soudris, "A novel methodology for temperature-aware placement and routing of fpgas," in *IEEE Computer Society Annual Symposium on VLSI*, 2007.

[11] W. Hung, C. Addo-Quaye, T. Theocharides, Y. Xie, N. Vijaykrishnan, and M. J. Irwin, "Thermal-aware ip virtualization and placement for networks-on-chip architecture," in *IEEE International Conference of Computer Design*, 2004.

[12] G. Link and N. Vijaykrishnan, "Hotspot prevention through runtime reconfiguration in network-on-chip," in *IEEE Design, Automation and Test in Europe*, 2005.

[13] A. Merkel and F. Bellosa, "Balancing power consumption in multiprocessor systems," in *ACM EuroSys*, 2006.

[14] S. Lopez-Buedo, J. Garrido, and E. Boemo, "Thermal testing on reconfigurable computers," *IEEE Design & Test of Computers*, 2000.

[15] H. F. Hamann, A. Weger, J. A. Lacey, Z. Hu, P. Bose, E. Cohen, and J. Wakil, "Hotspot-limited microprocessors: Direct temperature and power distribution measurements," *IEEE Journal of Solid State Circuits*, vol. 42, no. 1, Jan. 2007.