# FastSpot: Host-Compiled Thermal Estimation for Early Design Space Exploration

Darshan Gandhi, Andreas Gerstlauer, Lizy John
Electrical and Computer Engineering
The University of Texas at Austin
Email: {darshang, gerstl, ljohn}@utexas.edu

**Abstract**— Power and temperature of modern day systems have become important metrics in addition to performance. Static and dynamic power dissipation leads to an increase in temperature, which creates cooling and packaging issues. Furthermore, the transient thermal profile determines temperature gradients, hotspots and thermal cycles, which influence wear-out and reliability. Fast and accurate methodologies to facilitate estimation of temperature are needed during design space exploration. Traditional solutions rely on cycle-accurate simulations of detailed micro-architectural structures and are slow. We propose an approach that integrates accurate thermal estimation into existing host-compiled simulations. The developed methodology can incorporate different thermal models, such as HotSpot or discrete-time temperature evaluation models (DTTEMs). Using DTTEM as our thermal model, we show a 32000x increase in simulation throughput for temperature trace generation, incurring an average 0.06 K error in transient trace generation and 0.014 K error in steady state temperature measurements when compared to a cycle-accurate reference method. A comparative study between HotSpot and DTTEM based on speed/accuracy tradeoffs is reported for the thermal estimation process, showing that a 9x increase in simulation throughput can be achieved using DTTEM as the thermal model while incurring only marginal temperature measurement errors.

**Keywords**— Host-compiled simulation, back-annotation, power and thermal characterization

## I. **Introduction**

Power and temperature of modern day systems have become important metrics in addition to performance. In the early stage of design exploration, detailed information about power and thermal characteristics of the system is not available. The need for accurate and early estimation motivates the development of faster power and thermal simulations. Moreover, such simulations can provide quick tradeoff analysis during early design phase. Iterative and time consuming fine-tuning of designs across a range of optimization objectives and application benchmarks is often required. Hence, there is a need for a quick, accurate and integrated approach for thermal analysis.

A popular tool for temperature estimation is HotSpot [14]. HotSpot does not integrate power estimation and benchmark execution, but requires a previously collected power trace. The conventional way to estimate power is to execute the application on a cycle-accurate instruction-set simulator (ISS) that is integrated with a power estimation tool, such as Wattch [1] or that feeds ISS-based statistics into a power estimation tool, such as McPAT [9]. With benchmarks having billions of dynamic instructions, power estimation at the cycle-accurate simulation level is prohibitively slow. Moreover, HotSpot uses numerical analysis methods (Runge-Kutta), which slow down the transient temperature estimation process. The challenges involved are maintaining accuracy of results while providing high simulation speed. Light-weight and approximate models for thermal estimation can speed up early design space exploration significantly, albeit at the cost of accuracy. *Liu et al.* [10] explore periodicity in the power trace of a benchmark to provide a faster transient thermal simulation method. The temperature is calculated as cumulative effect of periodic responses of temperature and transient DC temperature response. *Han et al.* [7] propose TILTS, which utilizes discretization of input power traces in order to obtain a system of linear equations instead of integral equations. *Yang et al.* [16] develop ISAC for dynamic and steady-state thermal analysis. They use spatially and temporally adaptive techniques to reduce computation time in synthesizing complex ICs. These methodologies do not provide an integrated approach for thermal analysis and the input is primarily some existing power trace.

*Eratne et al.* [3] explore scheduling and floorplanning techniques for reducing peak temperatures (or hotspots) during the execution phase of a micro-processor. A multicore simulator, Multi2Sim, is integrated with McPAT [9] and HotSpot [14] in order to generate steady state temperature profiles of SPEC benchmarks to evaluate techniques for hotspot reduction. The run time or simulation speed of their approach is not reported. *Thiele et al.* [15] integrate thermal analysis in software synthesis for multi-processor systems. The system of differential equations for temperature evaluation is approximated to linear equations using a Discrete-Time Temperature Evaluation Model (DTTEM) approach [8] for small and constant time intervals. Parameters for thermal analysis are obtained before the task of mapping applications on resources, again using slow low-level simulation methods.

Our objective is to develop a methodology for fast, yet accurate thermal trace generation using fast simulation and fast thermal estimation approaches. Our work builds on the framework by *Chakravarty et al.* [2], which realizes a host-compiled back-annotation strategy for execution delay and total energy estimation during benchmark execution. Using a pre-simulation characterization phase, basic blocks in the intermediate representation (IR) of an application are annotated with latency and energy numbers obtained from cycle-accurate reference models. When the IR is compiled and executed on the host machine, they achieve very accurate timing and energy results at high simulation speeds. In our work, we extend this flow in order to estimate temperature for different benchmarks and obtain a transient temperature trace at high structural accuracy necessary for accurate thermal estimations. Our results are not directly com-

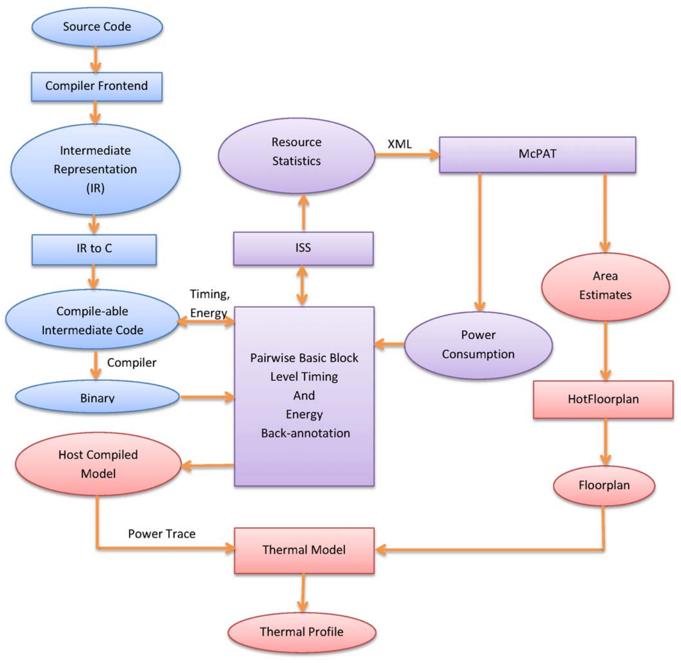


**Fig. 1:** Block diagram of FastSpot.

parable to [2], since temperature estimation is not considered in their work. We extend the approaches using simple models to compute power of the chip components at regular intervals from back-annotated target characteristics and thus provide a power trace. A temperature estimation model is then integrated with such a power trace to generate a thermal trace.

In the process of developing a host-compiled thermal modeling methodology, we make the following contributions:

1. We identify bottlenecks in a typical thermal profile generation process, which turn out to be cycle accurate simulation and resulting estimation of power traces for given benchmarks.
2. We propose an alternative and integrated methodology for fast generation of thermal profile, called FastSpot. It is developed on concepts of back-annotation and host-compiled simulation [5] [2], which provide a very high simulation speed at minimal loss of accuracy.
3. We integrate two different temperature estimation models in our methodology: HotSpot and DTTEM [15] [8], establishing the flexibility of the proposed approach.

The rest of the paper is organized as follows. Section II describes our proposed methodology for thermal analysis. Section III then elaborates on the thermal modeling approach we adopted within the overall host-compiled simulation strategy. In Section IV, we show results of validating our flow on three benchmarks and comparing results with the reference. Finally, the paper provides conclusions and directions for future work in Section V.

## II. Methodology

In this section, we explain the overall flow of the proposed FastSpot methodology. We briefly summarize prominent features of host-compiled simulation [2] that our work extends on. Fig. 1 shows the block diagram of our proposed approach.

The underlying concept of FastSpot is back-annotation at the intermediate representation level. Application source code is passed through a compiler to translate code into an IR that accurately reflects front-end optimizations. Following this, basic blocks (BBs) in the IR are annotated with their execution latency and energy, obtained using retargetable, cycle-accurate ISS and McPAT reference models, respectively. Details of this back-annotation process are described in [2].

During trace analysis of ISS output, a set of access statistics is collected and subsequently fed into McPAT. McPAT provides dynamic power dissipation of each component using the activity information. Finally, the power dissipated during a block execution is converted into energy and annotated in the IR code. This is called the characterization phase, which includes a pair-wise characterization of each block with all its possible predecessors in order to accurately account for path- and history-dependent effects on latency and energy.

We incorporate temperature estimation tools, such as HotSpot and DTTEM into our flow. Typically, thermal estimation tools require a form of power trace, which indicates power consumption in various components of the floorplan of a chip over time. We extend existing back-annotation approaches [2] such that the annotated energy dissipation values in the intermediate representation of the source code can be used by the host-compiled model to generate a structurally accurate power trace. In order to model the spatial distribution of temperatures, estimation tools also require the area and placement of components of a chip. Such area and placement estimates of various blocks are obtained using McPAT and the hotfloorplan utility. Combined, the power trace and the floorplan can be utilized by the integrated thermal model to generate the thermal profile over time.

## III. Thermal Estimation

A back-annotated intermediate representation of the source code, when integrated with the functional implementation of a thermal model results in FastSpot, a host-compiled model for temperature estimation. FastSpot achieves speedups due to faster power trace generation using host-compiled simulation and application of light-weight temperature models, such as DTTEM. In the following, we describe three aspects of our approach in more detail: (1) organization of the back-annotation process, (2) employed thermal models and (3) integration of thermal models into a comprehensive temperature simulation.

### A. *Organization of the Back-Annotation Process*

The back-annotation flow developed in [2] works on a block granularity to annotate information related to timing and energy. A mapping table is developed to map the extracted blocks from the binary to blocks of the compilable IR. Then, each block from the binary is characterized using the reference models. The important characterization information for the purpose of thermal analysis is latency of a block and energy consumption in all floorplan components. In our flow, the mapping table is augmented with component-specific information to enable correct back-annotation. The compilable IR is then appended with function calls at block-level granularity, which can access latency and energy values associated with unique block pairs. These

```
//Global variables being updated
unsigned long cycle_count;
//NUM_COMPONENTS is defined to be
//total number of components in a floorplan
double current_energy[NUM_COMPONENTS];
double previous_energy[NUM_COMPONENTS];
double Power_values[NUM_COMPONENTS];
unsigned long counter;

//The basic operations of functions remain the same
//irrespective of the thermal model
………………
void increment_latency(unsigned int n) {
        cycle_count += n;
        If (cycle_count > sampling_period*counter) {
                counter++;
                Power_values = calculate_power_array();
                //The following method depends on the thermal model
                calculate_temperature(Power_values);
                for (i = 0; i <NUM_COMPONENTS; i++) {
                        previous_energy[i] = current_energy[i];
                }
        }
}
void increment_energy(double m[]) {
        for (i = 0; i <NUM_COMPONENTS; i++) {
                current_energy[i] += m[i];
        }
}
```

**(a)** Global functions

```
………………
//Latency and energy_values are obtained from characterization phase
//The IR remains the same irrespective of thermal model
BB_i:
        //Appended functions
        increment_latency (latency_i);
        increment_energy (energy_i);
        //Block code
        j = j +1;
        end = (long unsigned int) p;
        if (j < 4) {
                goto BB_o;
        }
        else {
                goto BB_k;
        }
BB_k:
………………
```

**(b)** Back-annotated IR

**Fig. 2:** Organization of the back-annotated IR.

functions (Fig. 2(a)) increment variables that maintain running counts of total cycles elapsed and total energy dissipation per component as the back-annotated host-compiled binary gets executed on the host machine.

Fig. 2 (b) depicts a simplified structure of a back-annotated IR. The compilable IR contains optimized C code, separated at block boundaries and connected to other blocks according to the control flow graph of the application. The back-annotation process adds two function calls *increment_latency()* and *increment_energy()* to each block of the IR in order to update elapsed cycles and an array holding energy dissipation values for all components of the floorplan.

The augmented mapping table is responsible for associating latency and energy characterization data to the corresponding block. The back-annotation process thereby maintains predecessor-successor relationships through unique table entries and defines a global path-tracking variable as well as arrays to select characterization data of a specific block pair during runtime (not shown here).

The function *increment_energy()* simply increments a global array *current_energy[]* by the component-specific *energy_values[]* (obtained through characterization) of the current block. The function *increment_latency()* does similar updates for cycle counts. Moreover, the latter function is responsible for temperature estimation. Whenever elapsed cycles reach a multiple of the predefined sampling period, the function calculates energy dissipated in the previous sampling period by taking a difference of *current_energy* and *previous_energy* arrays. It also converts energy difference to power dissipation using knowledge of the sampling period length and operating frequency (function *calculate_power_array()*). The power values are used by *calculate_temperature()* in order to generate a temperature trace. This function is specific to the thermal model integrated into the flow.

## B. *Thermal Models*

In the following, we describe prominent features of the thermal models we integrated into the host-compiled simulation to generate transient and steady state thermal profiles, and evaluate speed/accuracy tradeoffs in the process.

### B.1 **HotSpot**

HotSpot [14] dynamically builds a representative RC model of the chip based on a given floorplan, and it computes the temperature profile of the chip over a sequence of time stamps (called sampling periods) based on a given trace of power dissipation values. HotSpot requires a structurally accurate power trace over all components in the floorplan at the chosen sampling period. We generate these traces using the extended back-annotation approach described in the previous section. HotSpot provides two modes for temperature estimation: using block-based or grid-based models. We choose the block model for computation of transient temperature profiles of a chip due to its advantage of higher speed during design space exploration.

### B.2 **Discrete-Time Temperature Evaluation Model**

DTTEM [15] [8] uses similar concepts as HotSpot for temperature estimation. This model requires conductance (G) and capacitance (C) matrices, which are characteristic of a chip floorplan. To maintain consistency between DTTEM and HotSpot thermal models, without loss of generality, we extracted RC representations from HotSpot and used MATLAB to generate the parameters required by DTTEM. Any assumption or approximations made during generation of RC models in HotSpot are thus retained by DTTEM, making the comparisons easier.

The primary difference between DTTEM and HotSpot is the temperature evaluation mechanism. DTTEM assumes that, with sampling of power values at small and constant time intervals, we can discretize transient temperature evaluation. This assumption leads to a simplified solution of the basic first-order heat transfer differential equation. We integrate this solution into our flow to achieve faster temperature estimation. The so-

```
//The function calculate_transient_temperature() captures model specific
operations.
//For the two models considered in our approach,

-----------------------------------------------------------------------------------------------

//HotSpot
void calculate_temperature(double *Power_values){
        create_hotspot_pipe(estimate_steady_state);
        write_power_array(Power_values);
}
-----------------------------------------------------------------------------------------------
//DTTEM
//NODES represent the number of junctions in RC  model of the floorplan
double T_coefficient[NODES][NODES];
double P_coefficient[NODES][NODES];
double P_current[NODES];
double T_current[NODES];
double T_next[NODES];
int      estimate_steady_state;

void calculate_temperature(double *Power_values){
        P_current = preprocess_power_values(Power_values);
        for (i = 0; i < NODES; i++){
                mmult = 0.0;
                for (j = 0; j < NODES; j++){
                        mmult += T_coefficient[i][j]*T_current[j] +
                                    P_coefficient[i][j]*P_current[j];
                }
                T_next[i] = mmult;
                T_current[i] = T_next[i];
        }
        print_temperature_array();
        if (estimate_steady_state == 1)
                generate_steady_state (Power_values);
}
```

**Fig. 3:** Organization of the thermal model.

lution to the differential equation is given as [15] [8]:

$$\mathbf{T[k+1]} = \mathbf{e^{-A\Delta t}} \cdot \mathbf{T[k]} + \mathbf{A^{-1}} \cdot (\mathbf{I} - \mathbf{e^{-A\Delta t}}) \cdot \mathbf{B} \cdot \mathbf{P[k]},$$

where $\mathbf{A} = \mathbf{C^{-1}} \cdot \mathbf{G}$, $\mathbf{B} = \mathbf{C^{-1}}$ and $\Delta t$ is the sampling time interval.

It can be observed that the temperature estimation is a linear function of the current temperature of the nodes in the circuit and the power dissipation in the time interval. Moreover, the exponential term $\mathbf{e^{-A\Delta t}}$ can be precalculated for a chosen sampling interval. Temperature for the next time interval can be calculated by considering two matrix multiplications and one matrix addition, reducing the evaluation time when compared to numerical approaches.

DTTEM can be used to estimate steady state temperatures in a chip as well. Solving the above equation with $\mathbf{T[k]} = \mathbf{T[k+1]}$ for all values of $\mathbf{k}$ under steady state assumptions yields

$$\mathbf{T_{SS}} = \mathbf{G^{-1}} \cdot \mathbf{P_{average}},$$

where $\mathbf{P_{average}}$ is the estimated power dissipation matrix in the steady state.

### C. Integration of Thermal Models

Fig. 3 represents two implementations of the thermal evaluation function for HotSpot and DTTEM, respectively. For a HotSpot-based thermal model, power values are then written to a Unix pipe (function *write_power_array()*) created at the start of execution of the host-compiled model. We have modified HotSpot to accept data streamed through a pipe and interpret the written power values to calculate the temperature. This process, repeated every sampling period, generates a transient thermal trace of the original application.

For a DTTEM-based thermal model, the evaluation function implements the necessary matrix operations to generate the transient temperature at the end of current sampling period. The coefficient matrices are obtained by dumping parameters of the RC network constructed by HotSpot. Other matrix related computations are done in MATLAB, and a header file defining the coefficients (matrices *T_coefficient[][]* and *P_coefficient[][]*) is generated. This is a one-time procedure for a given floorplan and consumes a small amount of time.

After the temperature computation, *T_next[]* is fed to the output file/screen and *T_current[]* is populated with *T_next[]* for the next iteration. The function *preprocess_power_values()* rearranges and populates other nodes of the RC model for which *Power_values[]* are not obtained in the host-compiled model. It should be noted that the number of nodes in the RC equivalent of the floorplan will be higher than the number of components, but the nodes acting as variable power sources (assuming point power sources at the center of the component) will be the same as the components.

The thermal model header file defines a variable *estimate_steady_state*, which can be used by the functional description of the model to evaluate steady state temperature. In the case of HotSpot, it simply adds an argument to the HotSpot run command for steady state estimations using the block-based model. For DTTEM, the variable enables steady state thermal evaluation (function *generate_steady_state()*) using the formula noted earlier.

Back-annotated function calls are an overhead to the original application binary and ultimately result in a slowdown compared to pure functional execution of the application. Therefore, the primary objective is to keep the function definitions simple and as small as possible. As described above, the functions do not contain many branch conditions (e.g. if statements) and the operations are mainly additions/multiplications. This reflects our objective to minimize overhead. Another issue in the HotSpot-based thermal model is a very high data transfer rate between the host-compiled model and HotSpot. The size of each data transfer depends on the number of floorplan components of the chip in consideration. Moreover, latency of the complete execution of the application and the sampling period dictates the number of such data transfers. Usage of files or print statements may slow down the thermal trace generation process considerably. Instead, we use pipe calls, such as popen/pwrite in order to minimize impacts of the operating system. A DTTEM-based approach requires no such transfers, and such problems are not encountered in a DTTEM-based thermal estimation process.

## IV. Experiments and Results

We have integrated our thermal estimation approach into the host-compiled back-annotation framework from [2]. Our FastSpot tool is available for download at [13]. To demonstrate the feasibility of our approach and to evaluate the tradeoffs, we apply the back-annotation flow to a simplified e200 Z6 (single

**TABLE I:** Details of MiBench benchmarks.

| Benchmark | Suite | Instruction count | Unique BB pairs |
|---|---|---|---|
| ADPCM | Telecom | 36,667,034 | 53 |
| CRC32 | Telecom | 12,319,886 | 7 |
| SHA | Security | 14,698,630 | 88 |

issue) and e200 Z4 (dual issue) PowerPC like architecture. Reference models for the architecture are provided by Freescale through their uADL [4] framework. We also incorporate the same enhancements to the model as described in [2] in order to accurately characterize block pairs. The chosen reference architectures can be classified as 32-bit processors having static branch predictors and no MMU, cache or FP units. Modeling of such components is considered future work. Although the uADL models do not include MMUs or caches, area estimates of McPAT and the floorplan generated by hotfloorplan model these components. The primary reason is to study a more generic floorplan from a temperature variation perspective. Accesses to these otherwise unavailable components are always zero. Hence, they do not contribute to dynamic power.

We created a reference flow (described next) for the purpose of validating and comparing our proposed methodology. These two methodologies are compared on the basis of simulation throughput, run time and relative accuracy of results. We choose benchmarks from the MiBench suite [6] [11] namely, ADPCM (Telecom), CRC32 (Telecom) and SHA (Security)- for comparison due to the highly pervasive nature of mobile computing. Benchmarks are compiled using gcc with O2 optimizations. Small input datasets are used and file I/O operations are converted to array operations. Dynamic instruction counts of benchmarks are obtained by executing them on the cycle-accurate reference ISS, and are listed in Table I. A custom application is also written in order to stress different architecture components and thus study variations in hotspot generation.

We perform our simulations on Z6 and Z4 like architectures with a floorplan area (for 90 nm process technology) of 4.77 $mm^2$ and 6.85 $mm^2$, respectively. The operating frequency is assumed to be 500 MHz and ambient temperature to be 318.15 K. We choose a sampling period of 10k cycles (unless stated otherwise) for a good tradeoff between precision and overhead in HotSpot as suggested by *Skadron et al.* [14]. Our methodology uses McPAT version 0.8 and the block-based model (unless stated otherwise) of HotSpot version 5.02.

## A. *Reference Flow*

The reference flow developed for comparison is similar to the one developed in *Eratne et al.* [3]. We construct a methodology that uses the uADL reference ISS from Freescale [4] to produce latency, energy and temperature traces. The front end in the reference flow is very straight-forward, where a cross-compiled binary is executed on the ISS and an output trace is obtained. Using similar scripts as in the back-annotation flow, we identify latency information and access statistics for different sampling intervals. For each sampling interval, McPAT is invoked to obtain power values, and the dynamic power trace is fed into HotSpot for temperature trace generation.

This flow is highly influenced by the dynamic instruction

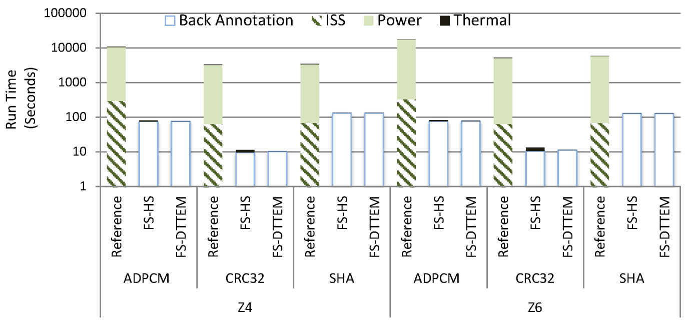


**Fig. 4:** Run-time breakdown of benchmarks.

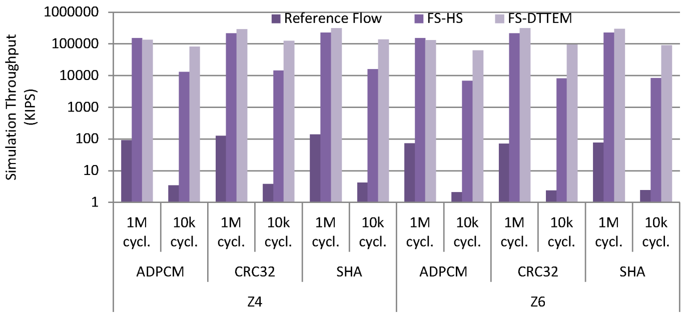


**Fig. 5:** Simulation throughput comparison for transient thermal profile generation at different sampling intervals.

count of the benchmark. As seen later, the majority of the run time is spent in power calculations by McPAT due to periodic and repetitive calculation of power at the granularity of the sampling interval.

## B. *Speed Evaluation*

Two metrics are considered for speed evaluations: (1) total run time and (2) simulation throughput. The reason is to quantify combined characterization and simulation time in FastSpot. For the reference flow, run time includes ISS execution time, total McPAT run time and HotSpot (HS) run time. For the FastSpot (FS) flow, the contributors to the run time are characterization time and execution time of the host-compiled model implementing the thermal model. For DTTEM-based FastSpot, we also include thermal parameter generation time in the calculation.

Fig. 4 shows the runtime breakdown of the reference flow and the two FS methodologies. The runtime improvements are similar in case of the Z4 and Z6 models. Overall, McPAT invocations contribute to a majority of the run time (ADPCM- 97.7%, CRC32- 97.8%, SHA- 97.9%) in the reference flow. By contrast, the major bottleneck in the proposed methodology is in the back-annotation time. It can be seen that for execution of a single benchmark with small inputs, time spent in generation and extraction of thermal parameters is insignificant and the choice of thermal model (HotSpot or DTTEM) does not play a major role. When compared to the run time of the reference flow, the run time of the FS flow in case of the Z6 model is 214, 503 and 43 times lower for ADPCM, CRC32 and SHA, respectively. In case of the Z4 model, run time of the FS flow is 125, 280 and 25 times lower for ADPCM, CRC32 and SHA, respectively. Time

**TABLE II:** Steady state simulation throughput.

| Benchmark | FS (DTTEM) | FS (HS-Block) | FS (HS-Grid) |
|---|---|---|---|
| ADPCM (Z4) | 136 MIPS | 115 MIPS | 99.1 MIPS |
| CRC32 (Z4) | 385 MIPS | 246 MIPS | 162 MIPS |
| SHA (Z4) | 298 MIPS | 207 MIPS | 154 MIPS |
| ADPCM (Z6) | 131 MIPS | 108 MIPS | 96.5 MIPS |
| CRC32 (Z6) | 373 MIPS | 176 MIPS | 124 MIPS |
| SHA (Z6) | 281 MIPS | 191 MIPS | 119 MIPS |

spent in characterization of the blocks of a benchmark depends on its control flow graph complexity. If there are a large number of unique block pairs in an application with low dynamic instruction counts, there is less improvement in total run time. The time spent in thermal parameter generation is fairly constant across these two architectures.

The simulation throughput is calculated as the ratio of the dynamic instruction count of a benchmark to the execution time of the host-compiled model. We do not count back-annotation time in the calculation of simulation throughput, as this is a one-time offline procedure for a given benchmark. After a benchmark has been characterized for all blocks, execution time of the annotated binary controls the simulation throughput, e.g. for repeated simulations over larger input data sets. Fig. 5 shows variations in the simulation throughput between two different thermal-aware simulations with sampling periods of 1 million and 10,000 cycles. Note that simulation throughput increases considerably with an increase in the sampling period. Therefore, the sampling period can be adjusted for higher throughput but less accurate temperature profiles.

For a sampling period of 10,000 cycles, average simulation throughput of the proposed methodology with a HotSpot model across both processors is 11 MIPS, whereas the simulation throughput for the reference flow is just 3.1 KIPS, making the former around 3600 times faster. When the comparisons are done for the proposed approach with DTTEM integration, the simulation throughput is 98 MIPS (a speedup over the reference flow of 32,000x). Overall, a DTTEM model achieves about a 9x speedup over a HotSpot based thermal evaluation.

It is to be noted that FastSpot with DTTEM does not achieve a speedup with respect to HotSpot in the steady state (SS) estimation process. Table II summarizes SS throughput results. Simulation throughput is of the same order and does not improve significantly with a DTTEM-based model. The reason is that the steady state temperature measurement is not affected much by the dynamic instruction count of the benchmarks, but transient temperature trace generation is. In our FS methodology, the transient temperature generation is the primary bottleneck, and we propose using DTTEM to overcome the issue.

## C. Accuracy Evaluation

We evaluated relative accuracy of the back-annotation flow when compared to the reference flow in terms of latency and temperature. The absolute and percentage errors for latency match the results obtained in [2]. The average error in latency measurement is 1.81% for the set of three benchmarks, where the highest error is present for ADPCM (5.38%). Errors in latency measurements in the host-compiled simulation are the primary source of all temperature errors. Latency deviations get propagated into energy and temperature errors, since power and temperature are calculated from latency information. The loss in accuracy for host-compiled latency measurements is mainly because of not being able to exactly replicate pipeline states for a basic block when characterizing blocks only across immediate predecessors. In addition, as results will show, use of DTTEM leads to additional errors stemming from approximations in the temperature model itself.

For temperature estimation, we do not have a single number to compare as the output is a trace over time. We applied two metrics, (1) transient temperature error and (2) steady state temperature error, for evaluating temperature accuracy of the proposed flow. We calculate average values for these absolute errors using the following equations:

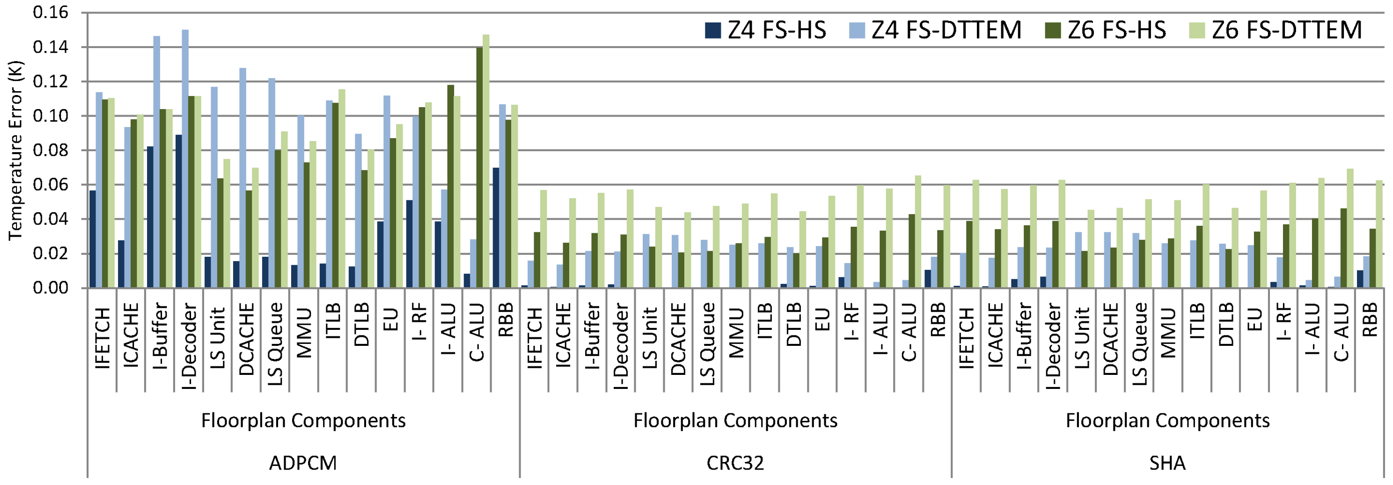
$$Average\ absolute\ transient\ temperature\ error$$
$$= \frac{\sum_{fp}\sum_{t}|T(fp,t)_{back\text{-}annotation} - T(fp,t)_{reference}|}{Number\ Components \times Trace\ Length}$$
$$\forall(fp) \in \text{Floorplan components}, \forall(t) \in \text{Time points}$$
$$Average\ absolute\ steady\ state\ temperature\ error$$
$$= \frac{\sum_{fp}|T_{ss}(fp)_{back\text{-}annotation} - T_{ss}(fp)_{reference}|}{Number\ Components}$$
$$\forall(fp) \in \text{Floorplan components.}$$

To calculate the transient temperature errors, temperature traces of the reference flow and the back-annotation flows are compared for all floorplan components, and the absolute differences in the values of transient temperatures are added up and averaged over the temperature trace length.

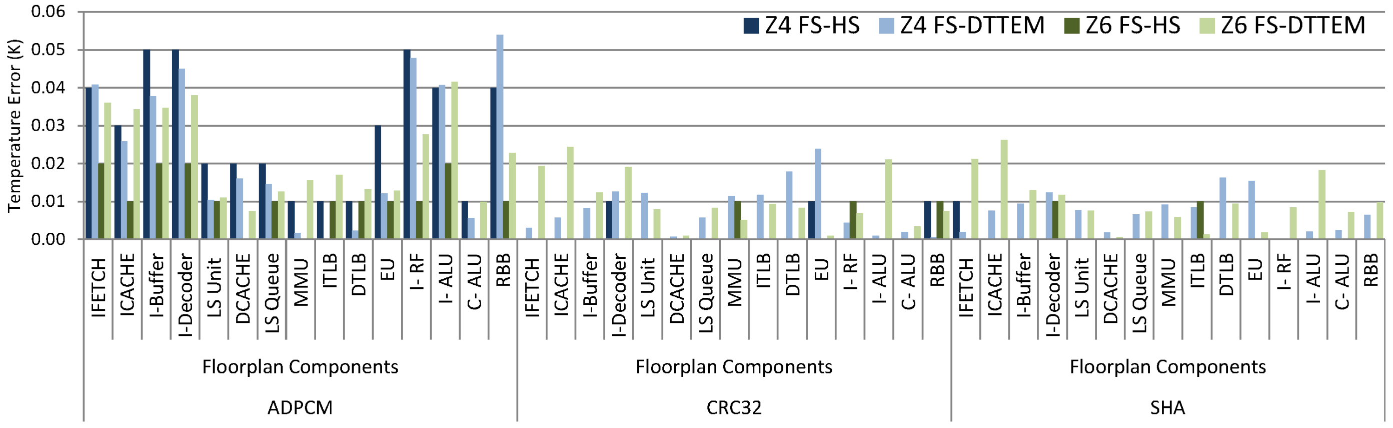
Fig. 6 (a) shows the average absolute transient errors per floorplan component for all benchmarks and processors. When HotSpot is used as the thermal model for the Z6 architecture, the maximum error is 0.14 K, whereas the maximum error in case of the DTTEM model is 0.15 K. Similarly, maximum errors in the Z4 case are 0.09 K and 0.14 K, respectively. The average error across all components of all benchmarks is 0.019 K higher than the HotSpot-based methodology in case of the DTTEM-based methodology for the Z6 processor. Similarly, the DTTEM-based methodology incurs a higher error (by 0.035 K) than the HotSpot-based implementation of FastSpot for the Z4 processor.

To calculate steady state temperature errors, HotSpot is configured to generate a temperature profile using the block-based model. HotSpot reports steady state temperature for all the components of a floorplan, which are compared against the values obtained from DTTEM. This metric is useful in practical analysis for studying temperature variation and hotspots. Fig. 6 (b) shows the average absolute steady state error for all benchmarks and processors. When HotSpot is used as the thermal model, the maximum error is 0.02 K for the Z6 processor and 0.05 K for the Z4 processor. The maximum errors in case of the DTTEM-based thermal model are 0.041 K and 0.054 K, respectively.

Table III summarizes the transient and steady state temperature errors over all floorplan components for both thermal models. In case of the HotSpot based FastSpot, the average transient



**(a)** Average absolute transient temperature errors.



**(b)** Average absolute steady state temperature errors.

**Fig. 6:** Component-level temperature errors.

error is 0.05 K for the Z6 and 0.014 K for the Z4 architecture. By contrast, in case of the DTTEM model, average transient errors are 0.07 K and 0.05 K, respectively. The average absolute steady state errors in both the models are low and usage of DTTEM instead of HotSpot increases the average steady state error only marginally for the Z6 model (0.014 K versus 0.0047 K for a 0.009 K increase). In case of the Z4 model, usage of DTTEM increases the average steady state error from 0.011 K to 0.013 K for a 0.002 K increase.

Maximum transient errors in a single sampling point range between 0.53 K and 0.75 K for a HotSpot and 0.55 K to 0.88 K for a DTTEM-based methodology. The highest observed chip temperature is 320.2 K for the Z4 and 319.6 K for the Z6 processor. Transient errors are higher in case of DTTEM mainly because of the approximations related to discretization. The solution to the heat transfer equation is approximated in the DTTEM-based approach. For example, it cannot capture the secondary effects of current temperature values on the next set of temperatures.

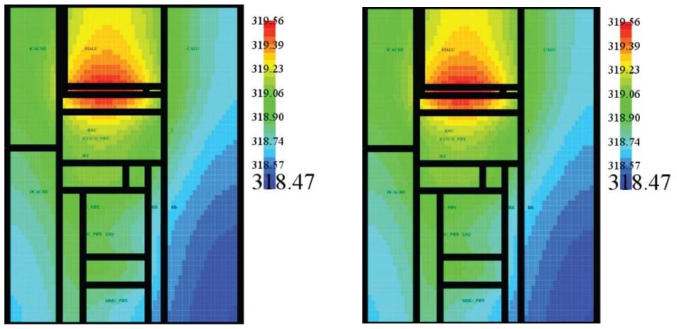
## *D. HotSpot Analysis*

In this section, we analyze steady state thermal profile generation using the reference flow and FastSpot using a grid-based model of HotSpot with a grid size of 64x64. The grid model of HotSpot divides the floorplan into layers of a grid and performs computation on small component cubes.

**TABLE III:** Average aboslute temperature errors.

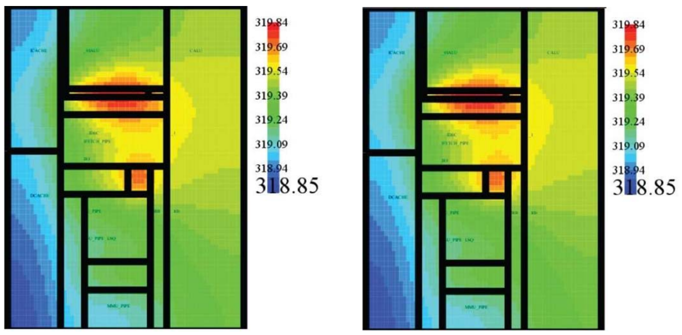
| Benchmark | Transient | | Steady state | |
|---|---|---|---|---|
| | FS-HS | FS-DTTEM | FS-HS | FS-DTTEM |
| ADPCM (Z4) | 0.037 K | 0.1 K | 0.029 K | 0.024 K |
| CRC32 (Z4) | 0.0017 K | 0.02 K | 0.002 K | 0.008 K |
| SHA (Z4) | 0.002 K | 0.022 K | 0.0007 K | 0.0072 K |
| ADPCM (Z6) | 0.09 K | 0.1 K | 0.011 K | 0.022 K |
| CRC32 (Z6) | 0.029 K | 0.053 K | 0.002 K | 0.01 K |
| SHA (Z6) | 0.033 K | 0.057 K | 0.001 K | 0.01 K |

Different benchmarks have different characteristics and may produce dissimilar temperature profiles when executed on a micro-architecture. We include the thermal profile of SHA (Fig. 7) and a manual stressmark (Fig. 8) on the Z6 architecture. Due to space limitations, we only show the profile for SHA, but results for the ADPCM and CRC32 benchmarks are similar. The manual stressmark focuses on Complex ALU operations (Mul/Div), which result in two hotspots at the instruction fetch block and the RBB (Result Broadcast Bus). We can observe that our FastSpot flow accurately tracks temperature variations across different benchmarks and that there are only minimal differences in steady state temperature profiles. Furthermore, it does so at significantly reduced runtime (see Table II).



**(a)** FastSpot with HotSpot-Grid **(b)** Reference methodology

**Fig. 7:** Hotspot formation in SHA (Z6).



**(a)** FastSpot with HotSpot-Grid **(b)** Reference methodology

**Fig. 8:** Hotspot formation in manual stressmark (Z6).

## V. Summary, Conclusions and Future Work

In this paper, we presented a novel simulation methodology for fast and accurate thermal estimations. This method uses host-compiled simulation instead of conventional ISS-based models. During a characterization phase, the source code of an application is annotated at block granularity with estimates, such as latency and energy consumption. Upon execution, the annotated binary creates a power trace and feeds the same into the thermal model. The thermal model operates on this trace and a floorplan to provide a final thermal trace and profile. The flexibility and integration capability of our methodology is demonstrated by integrating two different thermal models, HotSpot and DTTEM.

Using DTTEM as the thermal model, we achieve a simulation throughput of 98 MIPS, which represents a 32,000x speedup compared to a traditional flow for combined latency, power and temperature characterization. The error in measurement of temperature is very low. We report a 0.06 K and 0.014 K average absolute error in transient and steady state temperature generation, respectively. The DTTEM-based FastSpot methodology has major speed benefits, which come at the cost of increased measurement error when compared to the HotSpot-based implementation (which is around 9 times slower). We report a 0.02 K increase in transient temperature error and a 0.009 K increase in steady state temperature error for the Z6 model. Similarly, transient and steady state errors increase by 0.035 K and 0.002 K, respectively for the Z4 model. The sampling period can play an instrumental role in reducing the effects of approximations in the DTTEM model. Results demonstrate the consistency and accuracy of our flow for temperature measurement when compared to the reference flow, which indicates its viability for generation of thermal profiles during early design space exploration.

The FastSpot methodology spends a majority of the execution time in running HotSpot for temperature trace generation. We therefore integrated a faster and approximate thermal model (DTTEM) in the approach. We plan to further investigate implications of other fast thermal estimation methods [7] [10] [12] [17]. Evaluation of simulation bottlenecks can be beneficial for further optimizations. For example, fine-tuning the annotated code being added to the source code can reduce overhead and improve simulation throughput. Finally, we plan to investigate improvements in accuracy and scope, e.g. through modeling of dynamic components, such as MMUs, branch predictors or caches that are currently not considered.

## VI. Acknowledgement

This research was supported by the Semiconductor Research Corporation (SRC) Task ID 2317.001.

## References

[1] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *International Symposium on Computer Architecture (ISCA)*, 2000.

[2] S. Chakravarty, Z. Zhao, and A. Gerstlauer. Automated, retargetable back-annotation for host-compiled performance and power modeling. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2013.

[3] S. Eratne et al. Reducing thermal hotspots in multi-core processors using dynamic core scheduling. In *International Conference on Computer Design (CDES)*, 2011.

[4] Freescale. ADL release 2.0.0. `http://opensource.freescale.com/fsl-oss-projects`.

[5] A. Gerstlauer. Host-compiled simulation of multi-core platforms. In *IEEE International Symposium on Rapid System Prototyping (RSP)*, 2010.

[6] M. Guthaus et al. Mibench: A free, commercially representative embedded benchmark suite. In *IEEE International Workshop on Workload Characterization*, 2001.

[7] Y. Han, I. Koren, and C. M. Krishna. TILTS: A fast architectural-level transient thermal simulation method. *Journal of Low Power Electronics*, 3(1):13–21, 2007.

[8] A. Krum. *Thermal Management. In F. Kreith, editor, The CRC Handbook of Thermal Engineering*. CRC Press, 2000.

[9] S. Li et al. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009.

[10] P. Liu et al. Fast thermal simulation for architecture level dynamic thermal management. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2005.

[11] MiBench version 1.0. `http://www.eecs.umich.edu/mibench/source.html`.

[12] J. Nayfach-Battilana and J. Renau. SOI, interconnect, package, and mainboard thermal characterization. In *International Symposium on Low Power Electronics and Design (ISLPED)*, 2009.

[13] Retargetable Back-Annotator (RBA), version 1.1. `http://www.ece.utexas.edu/~gerstl/releases/`.

[14] K. Skadron et al. Temperature-aware microarchitecture. In *International Symposium on Computer Architecture (ISCA)*, 2003.

[15] L. Thiele, L. Schor, H. Yang, and I. Bacivarov. Thermal-aware system analysis and software synthesis for embedded multi-processors. In *Design Automation Conference (DAC)*, 2011.

[16] Y. Yang et al. ISAC: Integrated space-and-time-adaptive chip-package thermal analysis. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 26(1):86–99, 2007.

[17] A. Ziabari et al. Fast thermal simulators for architecture level integrated circuit design. In *IEEE Semiconductor Thermal Measurement and Management Symposium (SEMI-THERM)*, 2011.