Evaluating Linear Regression for Temperature Modeling at the Core Level

Dan Upton and Kim Hazelwood
University of Virginia

ABSTRACT

Temperature issues have become a first-order concern for modern computing systems. There are several approaches for dynamic thermal management, including reacting based on some threshold temperature, predicting based on history, or estimating localized temperatures based on performance counters. One possibility for more proactive management is to predict temperatures based on the upcoming instruction stream or performance counters. A logical approach is to use linear regression to generate a model based on the instruction stream, performance counters, or a combination of the two to predict temperatures at the next time step. However, we show in this paper that linear regression is unsuitable for predicting temperatures at the core level. This is primarily due to the fact that temperature at a time step is dependent upon both the processor activity and the temperature at the previous time step. As a result, application phases have long periods with similar instruction streams and performance counter values per time step but with different temperatures, which leads to prediction errors elsewhere in the phase. Incorporating temperature history as an input to the regression essentially leads to a last value predictor, which predicts the temperature will always be the same as the last time step. Thus, neither approach is truly suitable for predictive thermal management.

1. INTRODUCTION

Improvements in transistor technology and increasing clock rates have continually led to an increase in power density and core temperatures. High temperatures lead to several concerns. Higher heat dissipation tends to lead to increased cooling costs, and limits on practical cooling solutions impose an upper bound on reasonable heat dissipation for processors. Higher temperatures also lead to concerns about reliability and correctness. For instance, some processors set a bit indicating that correct execution is no longer guaranteed above a particular temperature [7], or may shut down the processor completely as a protection mechanism. Furthermore, even if the processor never exceeds its critical temperature, it has been suggested that a 10-15°C increase in operating temperature can decrease the processor's lifespan by as much as half [18], and 20°C cycling can increase the failure rate by 8x [19].

In addition to cooling solutions and automatic hardware throttling or protection mechanisms, there are many software-based thermal management solutions. Some of these rely on dynamic voltage and frequency scaling (DVFS) to control heat dissipation [1, 9, 16]. There are several possibilities for scheduling-based thermal management, such as changing the order of jobs based on some criteria [14], inserting idle loops to allow the processor to cool [10], or changing cores on a multicore processor [2, 4, 5]. DVFS-based solutions can also be combined with scheduling-based solutions [9].

Many of these software-based thermal management solutions are either static or are only able to use a limited amount of dynamic information. Static scheduling is much more practical in embedded systems or other instances where the workload is known and the thermal profile for all applications can be determined in advance. Most dynamic approaches are reactive, performing thermal management decisions when a threshold is reached, which limits the potential responses. Alternately, dynamic approaches may use limited temperature histories, while otherwise remaining agnostic to application-level characteristics.

Performance counters have been used to represent local activity for thermal management, either to avoid repeatedly using a functional unit to reduce the likelihood of hot spots [14] or to act as "soft sensors" instead of placing multiple physical temperature sensors across the die [3, 11]. However, thermal management decisions are generally made based on the temperature at the core level. Since core-level temperatures are related to the localized temperatures across the core, a logical next step is to use a similar approach of approximating core-level temperatures with multiple performance counters. Furthermore, only using performance counter values removes some higher-level information about the application, so another possibility is to instead model temperatures based on the application instruction stream. In either case, we may be able to predict the approximate upcoming activity, which would enable predictive management when coupled with a thermal model.

In this paper, we attempt to generate a linear regression model based on performance counters and a linear regression model based on two different representations of the instruction stream to predict core-level temperatures. We found initially that both of these approaches were prone to high maximum errors (up to 80%) and on average underpredicted or overpredicted temperatures by a few degrees. Since performance counters and the instruction stream represent different data – for instance, the instruction stream can not directly account for cache misses – we tried combining both sets of data. This slightly refines the predictions but still leads to a maximum error of nearly 60% and similar mispredictions by a few degrees. Since the error arises from the way temperature increases over time, with similar activity ultimately leading to multiple temperature readings, we tried adding historical temperature as an input to the model. However, this ultimately led to a noisy version of a last value predictor, which is unsuitable for predictive management because it will never predict changes before they happen.

The rest of this paper is organized as follows. Section 2 describes our experimental environment. Section 3 then discusses our various linear regression models and the problems associated with them. Section 4 describes related work. Finally, Section 5 concludes.

2. EXPERIMENTAL ENVIRONMENT

All of the experiments in this paper were carried out on an Intel Core 2 Duo with a 4MB L2 cache and 4GB of RAM. The machine was running CentOS 5.4 with a 2.6.33 kernel. The 2.6.33 kernel includes `perf events` [12] to access the hardware performance counters; we used `libpfm4` as an interface to `perf events`. Additionally, we configured the kernel with the `coretemp` driver, which allows reading the Core 2's thermal sensor [7].

In later sections of the paper, we will combine data from performance counters and the application instruction stream. We collected the instruction stream by instrumenting the application with Pin [13]. We recorded the instruction stream every 10 million instructions, which synchronizes well with performance counters by sampling performance counter values every 10 million instructions.

We took several steps to collect consistent temperature readings. First, we used the `cpufreq` kernel module to pin the processor frequency, which avoids any temperature changes during an execution due to automatic frequency variation. Fixing to a particular frequency does not specifically affect the applicability of our experiments; applications show similar temperature trends at different frequencies, just over different execution times and absolute temperature values. Second, we used the kernel's `isolcpus` flag to remove a given core from consideration by the scheduler, and the `sched_setaffinity` system call to pin the application of interest to that core. This guarantees that only the application we are profiling is using the core, which avoids any temperature or other hardware-level effects that would arise due to sharing a physical context. It also avoids changes in temperature due to migrating the application between cores. Finally, we let the processor idle for several minutes between executions, which removes the impact of one application's temperature on the beginning of the next application.

The Core 2 has one thermal sensor per core, described in the documentation as being over the hottest part of the core [7]. Access to the temperature sensor is essentially equivalent to reading a performance counter. It reports temperatures in integral degrees Celsius; this is one drawback to the sensor, as it tends to yield long periods at the same temperature and may also report full-degree oscillations due to rounding even if the actual temperature is changing by a much smaller amount. However, this should not significantly affect our insights, as it does not change that temperature is related to both activity and the previous temperature.

We perform our tests using a subset of the SPEC 2006 benchmark suite [6]: `perlbench`, `gcc`, and `astar` from the integer benchmarks, and `bwaves`, `dealII`, and `wrf` from the floating-point benchmarks. After observing the thermal profiles of all 29 benchmarks and 55 inputs in the reference set, we chose these benchmarks to provide a variety of application types, execution lengths, and thermal profiles. In particular, they have varying phase lengths and several variations in temperature. However, there are applications that essentially increase in temperature over time until they reach a steady state; in our benchmark set, some of the inputs to `perlbench` represent that class of application.

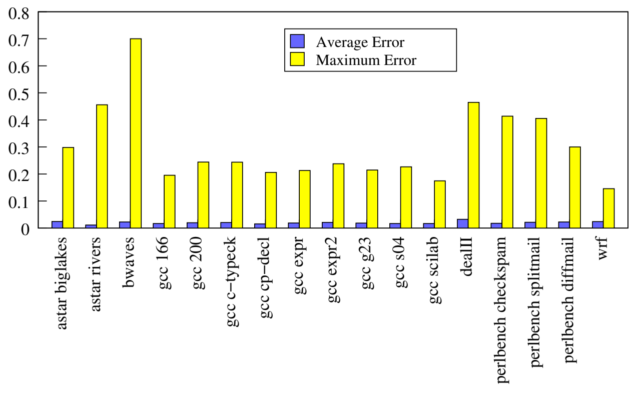

Figure 1: Average and maximum error for temperature prediction based on a linear regression from performance counters. Although the average error is generally 2-3%, the maximum error can be much larger.

3. LINEAR REGRESSION TEMPERATURE MODELING

This section will discuss each of the configurations we tested for linear regression modeling, along with a description of why each one is unsuitable for use in a predictive thermal management setting. We will begin with modeling based on performance counters in isolation. We will then consider modeling based on the instruction stream in isolation, using two different representations of the application instruction stream. Since performance counters and the instruction stream capture data at different levels, we then consider modeling based on a combination of the two data sets. Finally, we consider modeling based on a combination of performance counters, the instruction stream, and historical temperature data.

3.1 Performance Counter-Based Modeling

We collected a variety of performance counters from both the hardware and operating system levels to cover the various functional units. From the hardware, we started with standard metrics such as instructions, micro-ops, and floating-point ops per cycle, cache and TLB miss rates, and the branch misprediction rate. We augmented these with some lower-level performance indicators such as blocked loads, blocked stores, and cycles spent waiting on the memory bus. From the operating system level, we included minor and major page faults. Minor page faults indicate that the page is in memory but not in the page table and as such has a low performance penalty, while a major page fault requires loading the page into memory from disk. It is worth noting that while minor page faults were common, a few benchmarks suffered at most three major page faults and most experienced no major faults.

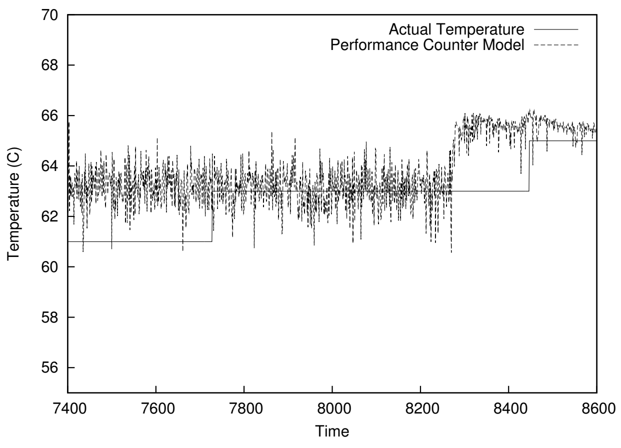

Figure 2: Actual and predicted temperature over time for a section of the first input to `astar`. The model predicts around a particular temperature for a phase, with oscillations due to noise from variation in the performance counter values.

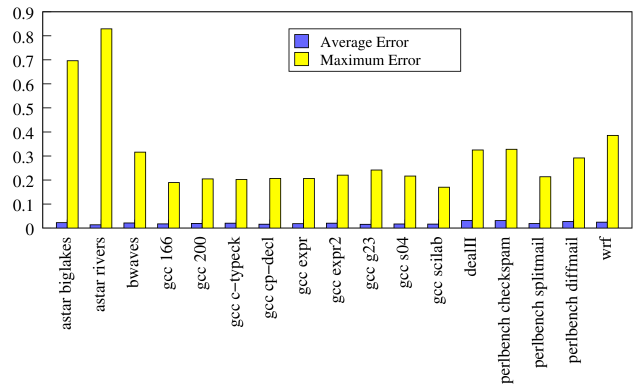

Figure 3: Average and maximum error for temperature prediction based on a linear regression from instruction categories. As with performance counter-based regression, the average error is 2-3%, but the maximum error can be much larger.

Figure 1 shows the average and maximum errors when predicting temperatures based on a linear regression model from performance counters. The average error is low, generally 2-3%. This amounts to predicting 1-2°C above or below the actual temperature. While the ideal case would be to perfectly predict the temperature, overpredicting by a small margin is unlikely to lead to severe performance penalties with predictive thermal management.

However, the maximum error is much larger, up to as much as 70% on `bwaves`. Such a large error is much more likely to lead to incorrectly throttling or descheduling an application. In practice, these large errors often happen primarily at the beginning of execution, when the processor is transitioning from an idle temperature to an active one. As a result, the model may significantly under- or over-predict temperatures. If such a method were being used in practice for predictive thermal management, a grace period could be given at the beginning of execution to avoid excessive throttling due to incorrect overprediction. While underprediction would not lead to throttling, a simple sanity check of predicting temperatures below the idle temperature of the processor would allow the management layer to ignore severe underpredictions.

Figure 2 compares the predicted and actual temperatures on the first input to `astar` over a period of roughly 10 billion instructions near a phase change. The phase change is clear near $x = 8270$, when the average predicted temperature and pattern of predicted temperatures changes. In this case, the model is predicting around the average temperature for a phase, with some variation due to the variation in counter values at each sample during the phase. This results in as much as a 4°C overprediction near time $x = 7600$, or a 3°C underprediction right before the phase change.

3.2 Instruction Stream-Based Modeling

We next consider modeling temperatures with the instruction stream in isolation. As noted in Section 2, we collected our instruction streams with Pin [13]. Pin is able to report the instruction mnemonic for each dynamically-executed instruction in the application. In addition, Pin's instruction decoder provides several logical categories that combine multiple instruction types. For instance, the decoder defines one category for conditional branches, one for unconditional branches, one for data transfer instructions, one for SSE, and so on. In total, the version of Pin used for this work defines 39 categories and recognizes 908 instruction mnemonics. This provides two different options for representing the instruction stream for the regression model.

Figure 3 first shows the average and maximum errors for predicting temperatures with a linear regression based on instruction categories. The results are similar to those for prediction based on performance counters, with an average error of 2-3%. The maximum error is slightly higher, just above 80% compared to 70% for performance counter-based regression. Again, in most cases, the largest errors occur at the beginning of execution as the processor is transitioning from its idle temperature to its active temperature range.

The largest error of 82% occurs on the second input to `astar`. In this case, the error occurs during a phase change near the end of execution. The actual temperature only drops one degree, but model spends two billion instructions swinging between over- and underpredicting the temperature.

Figure 4 shows the average and maximum error for predictions based on the instruction mnemonic. We omit `bwaves` and `wrf` due to memory errors when regressing the instruction mnemonic data. Compared to either performance counter or instruction category-based regression, both the average and maximum errors are reduced. The worst-case error is below 50%, and again occurs during the startup period of the benchmark. There are two possible reasons for the reduction in the error. First, there are many more variables involved, allowing the model to be more refined. Second, it is possible that different instructions or instruction sequences have different impacts on the temperature, but those variations are lost when aggregated into categories.

Figure 5 compares actual and predicted temperatures over time for the same period of the first input to `astar`. Recall that a phase change occurs near time $x = 8270$, which leads to an increase in temperature shortly afterward. The in-

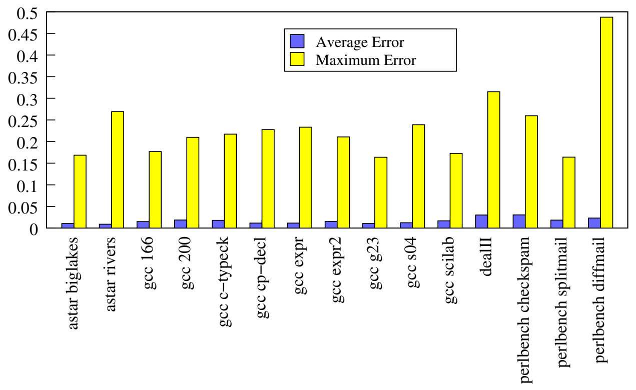

Figure 4: Average and maximum error for temperatures predicted by a linear regression on instruction mnemonics. Compared to the instruction category mix, the average and maximum errors are both reduced.

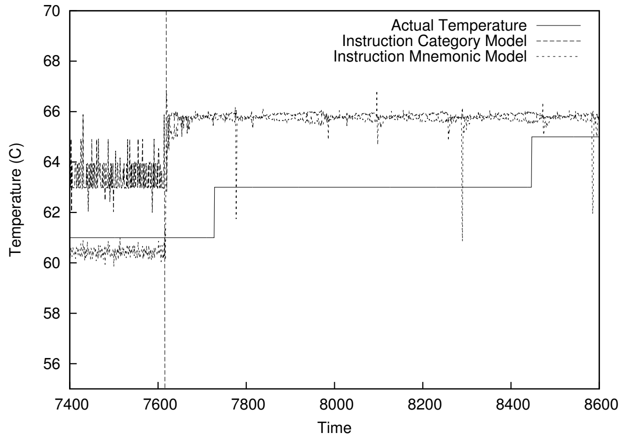

Figure 5: Predicted and actual temperatures over time for the two instruction stream-based methods. The instruction models predict the phase change and increase in temperature too early, which results in overpredicting temperatures by several degrees. Both models vary in predicted temperature, but the range of variation is lower for the instruction mnemonic model.

struction stream-based predictors react early to the phase change, leading to overpredicting the temperature for several degrees. In a predictive thermal management context, this may be acceptable, as predicting early provides plenty of lead time for making decisions, but overpredicting by a large amount or too early can lead to unnecessarily penalizing the execution. The comparison of predictions over time also shows the difference in prediction noise between the two models. Both models vary in temperature for a given average code sequence, but the predicted temperature with instruction mnemonics varies within a smaller range.

3.3 Combining Performance Counters and Instructions

We next look at creating a model using a combination of

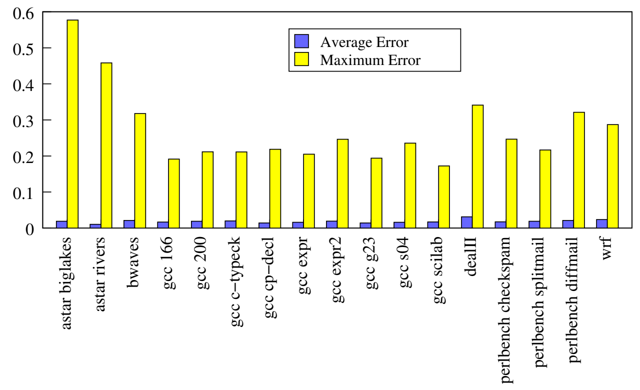

Figure 6: Average and maximum error for temperatures predicted by a linear regression based on a combination of performance counters and instruction categories. Both the average and maximum errors are reduced compared to a linear regression on either data set in isolation.

performance counters and instructions. Performance counters and the instruction stream provide different sets of information about application behavior. The instruction stream encodes higher-level information about the different ways in which hardware is being used, such as whether branches over a given time step are conditional or unconditional and direct or indirect. Performance counters encode lower-level information about hardware usage and effects that could either indicate increasing temperatures, such as higher activity and IPC, or decreasing temperature, such as a higher number of cache misses or pipeline flushes due to branch misprediction. Since the two data sets encode different information, a combination of the two may be able to refine the predictions.

Figure 6 shows the average and maximum errors for temperature prediction based on a combination of performance counters and instruction categories. The average error is reduced to less than 2% in almost all cases. Furthermore, the maximum error is decreased compared to modeling based on either data set in isolation. While performance counters and instruction categories in isolation led to maximum errors of approximately 70% and 80% respectively, the maximum error on the combined data set is 57%.

Figure 7 shows a similar comparison of the average and maximum errors for temperature prediction based on a combination of performance counters and instruction mnemonics. Again, the maximum error is reduced compared to using either data set in isolation, and the error for the combination with instruction mnemonics is less than that of the combination with instruction categories.

In both cases, we tested two combinations of the data. The first, shown in the previous two graphs, used both the performance counter values and the instruction stream for the time step being predicted. The second combination uses the instruction stream for the upcoming time step along with the performance counter values for the previous time step. This avoids requiring a phase predictor and would at worst miss phase transitions by one time step. In practice, the average and maximum errors were essentially the same for both configurations.

Figure 8 compares temperature over time for the instruc-

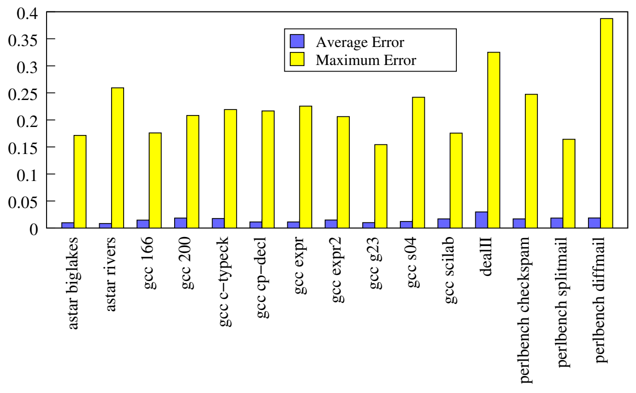

Figure 7: Average and maximum error for temperatures predicted by a linear regression on a combination of performance counters and instruction mnemonics. The average and maximum errors are reduced compared to a linear regression on either data set in isolation, and the error with instruction mnemonics is lower than that with instruction categories.

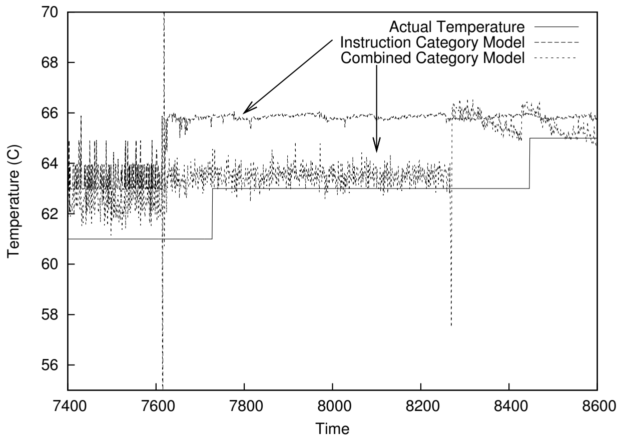

Figure 8: Actual and predicted temperatures over time for models using a combination of performance counters and instruction categories. The performance counter data prevents the phase change from being predicted too early, but increases the noise in the prediction after the phase change.

tion category model with and without including performance counter data on the same section of the first input to `astar`. The inclusion of the performance counter data slightly reduces the overprediction in one phase and prevents the instruction data from predicting the phase change too early. However, it increases noise in the predictions in the next phase. At other points in the execution, where the instruction mix suffers large errors but the performance counters do not, the combined model still has a large error but it is slightly reduced by the inclusion of performance counter data.

3.4 Modeling With Historical Temperatures

In all previous cases, one source of the error has been the

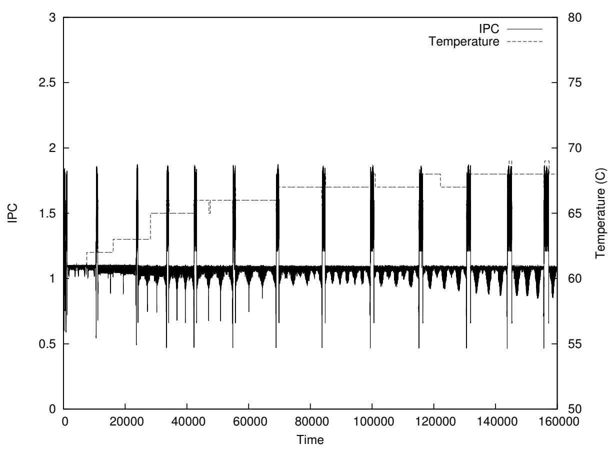

Figure 9: Temperature and IPC over time for the first half of `bwaves`. Although the code has a repeating phase with similar IPC and other characteristics, the code has a range of 7°C. This makes it difficult for a linear regression model to predict a correct temperature.

way temperature changes across an application phase or the way temperature changes across multiple repetitions of a given phase. Similar code is being executed, both from the standpoint of the instruction stream and the performance counter characteristics, but there are multiple temperature values. Consider for instance Figure 9, which shows temperature and IPC for the execution for the first half of `bwaves`. The same IPC, as well as code patterns and other hardware characteristics, repeats 12 times over the period shown. These repeating phases correspond to a range of 7°, which makes it difficult to map a particular input set to one “correct” temperature. This happens because the core temperature increases over time as more heat is generated than can be removed by the cooling solution. Also note that the rate of increase in temperature decreases as the temperature increases; the temperature increases by 5° over the first 40000 time steps, but only 2° over the rest of the period shown.

Since the current core temperature has an impact on how the code impacts further increases or decreases, a natural next step is to include the current core temperature as an input to the regression. Figures 10 and 11 show the average and maximum errors when predicting temperatures with a combination of performance counters, the instruction stream, and historical temperatures. Note that the y-axis now uses a log scale. This reduces the maximum error to approximately 1% and the average error to significantly less than that. Interestingly, the average error for using instruction mnemonics is slightly higher than for using instruction categories; while in previous cases, the mnemonic data provided more flexibility, in this case it seems to add more noise to the model.

Although the low errors here seem promising, we must also compare the predicted temperatures over time to see how such a model could perform in practice. Figure 12 compares the actual temperature over time to the combined predictor and a simple last-value predictor. Unfortunately, as the figure shows, the combined model is essentially acting as a noisy last-value predictor. This can be seen from the coeffi-

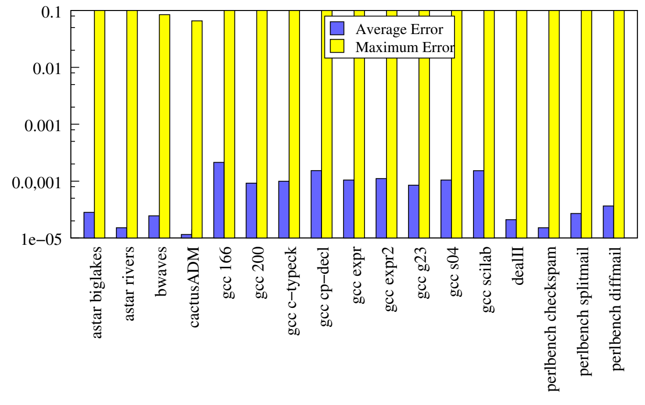

Figure 10: Average and maximum error for predicting temperatures with a combination of performance counters, instruction stream categories, and temperature history. Note the y-axis uses a log scale. By including all of the data points, we reduce the average error to less than one one-hundredth of a percent.

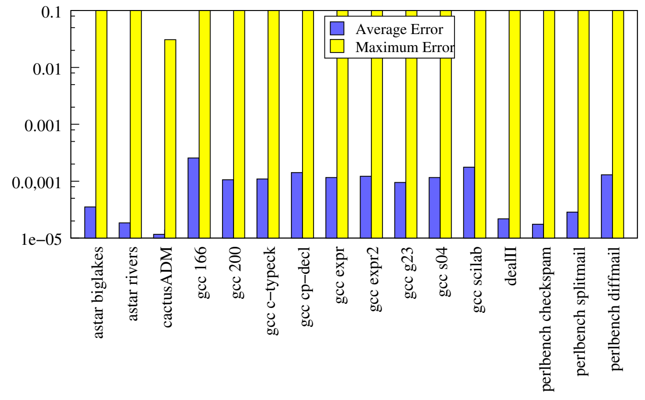

Figure 11: Average and maximum error for predicting temperatures with a combination of performance counters, instruction stream mnemonics, and temperature history. Note the y-axis uses a log scale. By including all of the data points, we reduce the average error to less than one one-hundredth of a percent. Errors with mnemonics are slightly higher than with instruction categories.

cients in the models, which assign a coefficient of roughly .99 to the previous temperature value. Since the last-value predictor will never capture temperature changes before they happen, it is unsuitable for predictive management.

3.5 Summary

This section has presented several potential methods for predicting the temperature of a whole core using linear regression based on application characteristics. Performance counters, which have been used to simulate localized temperatures, do not work well in combination for predicting temperatures at the core level. Instruction stream-based models also fail to suitably predict temperatures. A combination of performance counters and the instruction stream

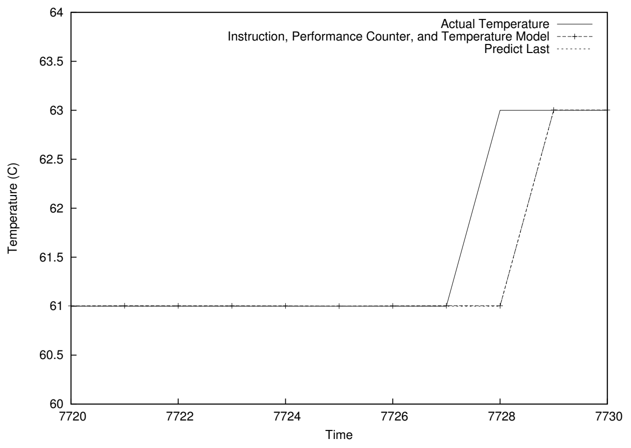

Figure 12: Comparison of the actual temperature and temperatures predicted by the combined model and by a last-value predictor. In the linear regression model, the previous temperature in the input is weighted so heavily that the combined model acts as a last-value predictor, which makes it unsuitable for predictive thermal management.

reduce the maximum error but still may overpredict or underpredict by several degrees. This is because similar code may lead to a wide variety of temperatures. However, including historical temperature as an input weights the model too heavily in the opposite direction, essentially leading to a last-value predictor.

4. RELATED WORK

Several works have modeled power and temperature based on performance counters. Singh *et al.* developed a power prediction model based on linear regression from four performance counters [17]. Isci and Martonosi predicted power from performance counters by deriving activity models for different processor components rather than using linear regression [8]. Unlike our temperature predictions, power measurements are not subject to historical effects, in that the power measurement at a given time is not a combination of the power drawn at that time plus power remaining from a previous time step. Lee *et al.* predicted localized temperature based on linear regression from performance counters [11]. Chung and Skadron modeled localized temperatures with a linear regression on performance counters and noted that using multiple counters per value sometimes reduced the accuracy [3]. These works differ from ours in that they did not attempt to model temperatures at the full core level. Merkel and Bellosa used task activity vectors, which approximate functional unit usage from performance counters, to roughly predict temperature and schedule to avoid hot spots [14]. Their work was again focused on localized temperatures. In addition, it focused on avoiding hot spots by not sequentially scheduling applications with similar activity vectors, rather than scheduling based on calculating specific temperature values.

Mesa-Martínez *et al.* noted that there was no correlation between temperature and IPC on its own [15]. Our work extends this notion to include a collection of performance counters, along with noting the historical temperature effects as one reason for the lack of correlation.

5. CONCLUSION

In this paper, we have shown that linear regression is an unsuitable approach for modeling full-core temperatures based on application-level characteristics. Linear regression initially seemed like a promising approach, as other works have successfully used single performance counter values to model localized temperatures. We started with full-core temperature predictions based on performance counters or the instruction stream in isolation and showed that although it had a low average error, it had a high maximum error and tended to mispredict by a few degrees. We then combined the two application representations, which noticeably reduced the maximum error but only slightly reduced the average error and misprediction range. After showing these mispredictions arose from the effects of temperature increasing over an application phase, we tried modeling based on a combination of application characteristics and temperature histories. When including temperature as a history, the linear regression model essentially became a last-value predictor, predicting temperature would remain the same as that at the previous time step, which is not useful for predictive management. As a result, linear regression based on application characteristics is not suitable for predictive temperature management at the core level.

6. REFERENCES

[1] B. Chen, W. P. T. Ma, Y. Tan, A. Fedorova, and G. Mori. GreenRT: A Framework for the Design of Power-Aware Soft Real-Time Applications. In *Workshop on the Interaction Between Operating Systems and Computer Architecture*, Beijing, China, June 2008.

[2] J. Choi, C.-Y. Cher, H. Franke, H. Hamann, A. Weger, and P. Bose. Thermal-aware Task Scheduling at the System Software Level. In *Symposium on Low Power Electronics and Design*, Portland, OR, August 2007.

[3] S. W. Chung and K. Skadron. Using On-Chip Event Counters for High-Resolution, Real-Time Temperature Measurements. In *Conference on Thermal and Thermomechanical Phenomena in Electronic Systems*, San Diego, CA, June 2006.

[4] A. K. Coskun, T. S. Rosing, and K. C. Gross. Proactive Temperature Management in MPSoCs. In *Symposium on Low Power Electronics and Design*, Bangalore, India, 2008. ACM.

[5] A. K. Coskun, T. S. Rosing, and K. Whisnant. Temperature Aware Task Scheduling in MPSoCs. In *Conference on Design, Automation and Test in Europe*, Nice, France, 2007.

[6] J. L. Henning. SPEC CPU2006 benchmark descriptions. *SIGARCH Computer Architure News*, 34(4), 2006.

[7] Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3: System Programming Guide*. Intel Corporation Order #253668-037US, January 2011.

[8] C. Isci and M. Martonosi. Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data. In *Symposium on Microarchitecture*, San Diego, CA, December 2003.

[9] R. Jayaseelan and T. Mitra. Temperature Aware Task Sequencing and Voltage Scaling. In *Conference on Computer-Aided Design*, San Jose, CA, November 2008.

[10] E. Kursun, C.-Y. Cher, A. Buyuktosunoglu, and P. Bose. Investigating the Effects of Task Scheduling on Thermal Behavior. In *Workshop on Temperature-Aware Computer Systems*, Boston, MA, June 2006.

[11] J. S. Lee, K. Skadron, and S. W. Chung. Predictive Temperature-Aware DVFS. *IEEE Transactions on Computers*, 59(1), January 2010.

[12] Linux. perf - Performance analysis tools for Linux, 2009.

[13] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. Janapa Reddi, and K. Hazelwood. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. In *Conference on Programming Language Design and Implementation*, Chicago, IL, USA, 2005.

[14] A. Merkel and F. Bellosa. Task Activity Vectors: A New Metric for Temperature-Aware Scheduling. *SIGOPS Operating Systems Review*, 42(4), 2008.

[15] F. J. Mesa-Martínez, E. K. Ardestani, and J. Renau. Characterizing Processor Thermal Behavior. In *Conference on Architectural Support for Programming Languages and Operating Systems*, Pittsburgh, PA, March 2010.

[16] L. Miao, Y. Qi, D. Hou, and Y. Dai. Energy-Aware Scheduling Tasks on Chip Multiprocessor. In *Conference on Natural Computation*, Haikou, Hainan, China, August 2007.

[17] K. Singh, M. Bhadauria, and S. A. McKee. Prediction-based Power Estimation and Scheduling for CMPs. In *Conference on Supercomputing*, Portland, OR, 2009. ACM.

[18] R. Viswanath, V. Wakharkar, A. Watwe, and V. Lebonheur. Thermal Performance Challenges from Silicon to Systems. *Intel Technology Journal*, Q3 2000.

[19] L.-T. Yeh and R. C. Chu. *Thermal Management of Microelectronic Equipment*. American Society of Mechanical Engineers, 2002.