Investigating the Effects of Task Scheduling on Thermal Behavior

Abstract — Thermal behavior is expected to be an important design consideration for the next generation of microprocessors. Most dynamic thermal management techniques are based on the principle of ‘reactive throttling’, where some form of performance throttling (such as voltage or frequency scaling) is invoked, after a pre-architected temperature threshold has been exceeded. As such, there is performance degradation with each such DTM technique. In some cases, if the temperature threshold is set to a conservative value for package/cooling cost considerations, the degradation can be quite severe. In this paper, we investigate the potential benefits of temperature-aware task scheduling with minimum or no performance overhead. We explored policies based on metrics such as average temperature; dynamic profiling of threads and localized heating, instead of IPC based metrics. Our preliminary results indicate that thermal-aware task scheduling provides significant alleviation on chip temperatures. We observed up to 52% reduction in the number of cycles above the thermal threshold, compared to a thermally-oblivious policy. On average, our MinTemp policy yields less than 3% thermally critical cycles, even on a challenging benchmark set such as SPEC2000 - with 12 of the 25 benchmarks above 360°C. As we employ already existing scheduling infrastructure, there was no considerable change in net performance.

1. Introduction

Thermal characteristics of modern microprocessors have presented numerous challenges in recent years. Continuous increases in power dissipation along with feature scaling caused the power density to double every 3 years [1]. The corresponding elevation in the on-chip temperatures have started to become a critical design constraint. Thermal characteristics of microprocessors affect vital aspects of the microprocessors: such as timing, reliability, fault-tolerance as well as increased packaging and cooling costs. As the reliability of an electronic circuit is exponentially dependent on the junction temperature: a 10-15°C rise in the operating temperatures results in~2x reduction in the lifespan of the device [7]. Furthermore, effective operating speed decreases at higher temperatures due to the temperature dependence of carrier mobility.

Static power dissipation, which is expected to constitute as high as 50% of the total power consumption [8] in post-90nm CMOS technologies, has exponential dependency on temperature. The positive feedback loop between leakage power and on-chip temperatures, also called as ‘Thermal Runaway’, can cause serious damage if not carefully controlled through the package/cooling and dynamic thermal management. A wide range of packaging and cooling techniques has been proposed to alleviate thermal problems. However, their costs increase at a super-linear rate (>$3 per Watt after 60°C, according to [5]) and package impedances are saturating to values that are challenging to manage cost-effectively. Due to the increases in the transistor count and clock frequencies, it is no longer practical to design the processor packaging for the worst-case temperature; at least, with regard to the desktop and lower-end server market. Many chip producers use packaging that targets a safe threshold below the absolute worst power/thermal behavior. For instance, Intel’s Pentium4 is reported to utilize packaging that is designed for around 80% of the worst-case power dissipation. Beyond this threshold the thermally challenging, yet infrequent, workloads are handled with global clock throttling [5].

Dynamic thermal management is commonly used to manage the gap between the absolute worst-case temperatures and packaging constraints. There has been a wide range of dynamic thermal management techniques proposed in recent years [2]-[9]. As the thermal problems exacerbate for the next generation microprocessors the need for efficient dynamic thermal management is also becoming more prominent. Most of the current DTM techniques use throttling of processor resources such as frequency, voltage or pipeline stages, in order to keep temperatures below a safe threshold. However, in some cases the corresponding performance degradation is significant, depending on the frequency and the severity of heating problems. For instance, activity migration is usually accompanied with considerable performance loss due to the overhead of moving processor state and the cold start effects [12].

Instead, if one can leverage the existing context swap infrastructure supported by the system software (e.g. the operating system and hypervisor layers) and implement a temperature-aware task scheduling heuristic, temperature reduction without additional on-chip hardware complexity is possible. The goal of this study is to investigate the potential benefits of scheduling techniques for temperature management with minimum or no performance degradation. Our initial experimental analysis indicates that thermal-
aware scheduling policies alleviate the on-chip temperatures. The policies successfully manage to keep the temperatures within a safe range, even for challenging benchmark sets such as SPEC2000 on a Power4-like architecture, where 12 of the 25 benchmarks have temperatures above 360°K. The scheduling policies we investigated in this study are implemented on single core architecture. Yet the same concept can be applied to a multi-core processor. Moreover, the increase in the number of threads and cores is likely to reflect as an advantage that scheduling policies can utilize even further.

It is important to note that temperature-aware operating system scheduling is beyond the scope of this paper. Our goal is to probe the limits of thermal-aware task scheduling through the use of architectural tools and techniques. The complete implementation of temperature-aware scheduling policies in the operating system kernel is a part of our future work.

Rest of the paper is organized as follows: In Section 2 we discuss the most relevant related work; Section 3 presents a motivating example; preliminaries on temperature modeling in Section 4, followed by a discussion of experimental methodology in Section 5. Section 6 introduces the proposed thermal-aware scheduling policies and finally concluding remarks and future work are in Section 7.

2. Related Work

Operating system scheduling policies for temperature management has been studied by various researchers in the past. In this section we only discuss the closest studies. In [6], Moore et.al. investigated temperature-aware workload placement in data centers. They analyze the thermodynamic behavior specific to the data center, and propose heuristics for improvement. In [15], Ghiasi et.al. analyzed a number of temperature-aware operating system scheduling policies for asymmetric cores. The most relevant related work to our study is [13]. In this work, Gomaa et. al. present temperature-aware thread assignment and migration policies for a multi-core SMT scenario. Their results are very valuable in assessing the potential of thread assignment on processor temperatures.

The differences between the previous work and this study also highlight our main motivation and contributions:

- In this study we explore a single thread single core scenario where thread migration is not a possible solution. Hence the goal is to investigate the limits of thermally effective thread assignment policies.

- Microprocessors time constants are critical in determining the feasibility of different temperature management schemes. The thermal time constant values are determined by a combination of architectural block properties such as thermal capacitance, resistances, layout area, as well as thermal characteristics of silicon and the process technology. The time constants can be quite different for different processors as a result of these factors. In [13] thermal time constant of the microprocessor is reported to be 10msec. In this kind of scenario, rapidly heating and cooling the processor is possible, since the time constant guarantees that a heated block will be below the thermal threshold in few milliseconds. During our experimental analysis, we found the time constant for Power4-like architecture to be much longer, in the 100 msec range. We observed up to 200 msec time constants for architectural blocks. Therefore, performance degradation due to cooling time of a processor block is much more prominent in our case. This fact motivates our attempts to manage the temperatures so that threshold is not reached. Exceeding the threshold is bound to cause degradation due to elongated throttling periods. Yet another effect of our longer time constant is the feasibility of slowly tuning the on-chip temperatures with each thread. As 10 msec time slices per thread constitutes a small percentage of the thermal time constant range in 100 msecs each thread can slowly tune the processor temperatures to a desired target.

- Previous studies such as [9] and [12] reveal that thread migration can yield significant performance degradation, caused by the copying of register state between cores and the cold start of caches. According to [9] thermally-triggered activity migration causes as high as 53% performance degradation on a two-core monolithic architecture. In this study our goal is to eliminate the loss from DTMs, minimizing and even eliminating the degradation from costly techniques such as migration. In order to achieve this goal we propose using the already existing task scheduling infrastructure. Our experimental analysis illustrates that MinTemp policy achieves limiting the thermally critical cycles to only 3% with virtually no performance degradation.

- We also investigated alternative metrics to Hi-IPC/Low-IPC, INT/FP, and IPC-based resource usage of [13]. IPC based metrics have shortcomings such as:
  - Limiting the negative effects of thermal problems to performance loss. In reality, other factors such as fault-tolerance, reliability, packaging and cooling costs are also affected by temperature profile.
  - Spatial distribution of hotspots on the chip is not taken into consideration. The proximity of hotspot plays an important role in exacerbating the temperatures.
To better handle these considerations we use real-time temperature readings of microprocessor blocks, along with dynamic thermal profiling of threads in the scheduling queue. Finally, our main goal is to finely tune the processor heating with each scheduled thread so that such heating scenarios can be minimized.

3. Illustrative Case

Let us consider a scenario where a thread from benchmark Mgrid has been running on a Power4-like architecture and the steady state temperatures have been reached. Mgrid has 367°K hotspot temperature without any dynamic thermal management. It causes floating point register file (FPU_REG) to exceed the temperature threshold of 358°K. For current microprocessors the scheduler assigns a different thread on the processor based on criteria other than temperature, usually in a form that is a variation of round-robin policy.

In this case, let us consider the presence of a monitoring facility that collects thermal profile of the threads in the scheduling queue. Furthermore, this unit collects the temperature readings of the processor blocks from on-chip sensors at the end of the time slice (10msec in this case). This information is passed on to the scheduler, where an informed decision can be made. Just as the FPU_REG is about to reach the temperature threshold, the scheduler can select the next thread from the thread queue based on the thermal profiles and the heating information from the on-chip sensors. Let us also assume that the threads extracted from SPEC2000 benchmark set are available in the scheduler queue. Since we look at the limit case where our starting point was slightly below 85°C, benchmark selection is crucial and the scheduler has a number of choices:

**Scenario 1:** One possibility is to select threads oblivious of the temperature profiles and schedule Applu in the next time slice. Applu has a steady state temperature of 365°K for FPU_REG. Hence the heating problem will persist and exacerbate above 358°K. A hardware DTM technique such as fetch throttling, dynamic voltage scaling or clock throttling has to be activated for the core to cool down below the safety threshold.

**Scenario 2:** It is also possible to select a low-temperature benchmark such as Mcf, which has the lowest thermal profile of the traces we experimented on. In this case, core temperature decreases and no DTM is needed.

**Scenario 3:** Alternatively, the scheduler can select yet another high-temperature benchmark such as Crafty, which reaches 363°K for FXU_REG. However, since the heating patterns are complementary, the FPU_REG slightly cools down without exceeding the threshold and the FXU_REG sees a slight increase in temperature.

An important point to note is that in none of these scenarios will there be a very significant temperature change: Neither Crafty reaches 363°K at the end of the next cycle, nor Mcf presents a dramatic decrease in temperatures. The temperature change for each benchmark will be incremental. This is caused by the fact that the thermal time constants of the processor are significantly longer than the scheduling time slices. The scheduling infrastructure already exists on the system and the thread assignment takes place at pre-determined time slices. Hence temperature-awareness provides the opportunity to finely tune the on-chip heating with each thread scheduled. We discuss this in more detail while presenting our temperature-aware scheduling policies in the following sections.

4. Preliminaries

Temperature modeling for microprocessors is based on the duality of the electrical and thermal phenomena. On-chip heating and cooling acts according to the exponential RC curves represented by the time constant. In electrical analogy, RC time constant is defined as the time required to charge a capacitor to 63.2% of the final value; or equivalently, the time required to discharge the capacitor to 36.8% of the initial value. Thermal transient behavior of an architectural block can be similarly estimated with the corresponding thermal time constant.
The time constant for an architectural block or floorplanned unit is dependent on the physical properties such as the thermal capacitance and resistance; which are dictated by the physical dimensions of the block and the intrinsic thermal properties of the silicon die material. It is independent of the power dissipation. A thermal model derived from University of Virginia’s HotSpot [4] built on top of the energy models within the Turandot/PowerTimer simulator [10], [11] were used to estimate the temperature characteristics in our experimental analysis.

Figure 1 displays the exponential decay curve for benchmark Mcf, when the active power dissipation is reduced to 0 Watts, as the benchmark stops executing. The x axis represents temperature samples. Each sample is taken at 100K cycles hence the x-axis of the figure illustrates that RC time constants are in 100msec range. Thermal time constant is independent of the power dissipation of the architectural block and program behavior. We have observed thermal time constants in the range of 80msec to 200msec for various architectural blocks.

4.1. SPEC2000 Temperature Profile

Initial thermal analysis of the traces from SPEC2000 benchmark set was performed using Turandot power performance and thermal simulator. We used 400 million and 1 billion instruction traces. Figure 2 illustrates the maximum temperatures architectural blocks over the 400 million traces. SPEC2000 is a thermally challenging benchmark set with 12 out of 25 benchmarks has blocks with temperatures above the 360°K. Keeping the temperatures below 358°K temperature threshold without a dynamic temperature management scheme is a challenging task over such a pool of threads.
Figure 2. Thermal profile of the traces from SPEC2000 benchmark set.

The average temperature of the traces is 351.2°K. On average the temperature profile of SPEC2000 on a Power4-like architecture is closest to Gcc, which has average temperature of 351°K and similarities hold in terms of heating for individual blocks as well. In general IDU (Instruction decode unit), FPU_REG (Floating Point Register File), FXU_REG (Fixed Point Register File), FXU_B1 (Fixed Point Unit) are the thermally challenging blocks. According to our experimental results on traces SPEC2000 has limited thermal variation within phases of individual benchmarks. For SPEC2000 traces of both 400 million and 1 billion instructions, the maximum deviation between maximum and average temperatures was less than 5°C on a Power4-like architecture.

5. Methodology

For our experimental analysis we used PowerPC simulator, Turandot [10] which utilizes HotSpot [4] models for temperature analysis of Power4-like microprocessor architecture. Each microarchitectural block is represented by a node in the analogous RC network, where the nodes are connected to each other with corresponding thermal resistance and capacitances. Similarly power density for each block is generated by Turandot’s built-in energy models [11]. The leakage power dissipation is taken into consideration along with the positive feedback loop between leakage power and temperature. Then, the corresponding differential equations are solved by 4th order Runge-Kutta method [4] during the HotSpot simulations. We assume process technology 180nm, clock frequency 1.1GHz, and supply voltage 1V with for our experimental analysis. We used an ambient temperature of 45°C, initial temperature 60°C, thermal threshold for DTM 85°C based on ITRS data [8] and packaging and cooling characteristics.

As mentioned earlier we used traces generated from SPEC2000 benchmark set 400 million instructions. After the initial power and temperature analysis of the traces a simulated scheduler keeps track of the temperatures and initiates Turandot simulations for each time slice. Thread scheduling is based on Round-robin policy for fairness accompanied with additional temperature-aware policies. Each thread is given 10 msec time slices before being interrupted and replaced by the next thread dynamically selected by the policy based on the on-chip temperatures at the end of that particular time slice.

6. Temperature-Aware Scheduling Policies

We have implemented and investigated a number of task scheduling policies. We assume that our baseline architecture is equipped with thermal sensors where the block temperatures can be read at the end of each scheduling time slice. This information is passed on to the scheduler so that the next thread selection can be based on the heating patterns observed in the current scheduling window (or time slice). We used the default time slice length of linux, 10msec, for our experimental analysis.

As the execution of threads is in round robin fashion this profiling information can be acquired dynamically at the first time slice of the scheduled task. During the next time slice, the scheduler has the estimated temperature behavior of the task and can schedule intelligently based on this information. Also, in some cases, a task can lose its time slice prematurely, if a pre-specified temperature schedule has been exceeded. According to our experimental results on traces SPEC2000 has limited thermal variation within phases of individual benchmarks. For SPEC2000 traces of 400 million and 1 billion instructions, the maximum deviation
between maximum and average temperatures was less than 5°C. This shows that the thermal profile of a thread is a good enough indicator of the thermal behavior of the thread next time it is assigned to a core. The results presented in the experimental analysis section are for initial profiles of the traces, yet the dynamic profiling results are consistent.

It is worth noting that temperature threshold is a function of silicon thermal behavior as well as the packaging and cooling characteristics of the processor. We used a critical thermal threshold value of 85°C and a safety threshold value of 82°C during our experiments for a Power4-like architecture. The rest of the temperature values and benchmark characteristics in this study are also for the same infrastructure. The temperature characteristics of threads and architectural blocks are expected to vary on other processors, yet the scheduling results still hold.

In this section we discuss a number of scheduling policies and compare their effectiveness in maintaining on-chip temperatures below the pre-set temperature threshold. For sake of comparison, we assume that all policies start with steady state temperatures of Mgrid. The first benchmark selection at the start point is also set to be Ammp for similar reasons. Furthermore, for clarity of the display, the figures present results for the temperature behavior of the corresponding policy on a limited number of SPEC runs. Notice that this also limits the policies in the number and variation of available threads; in a less restricted pool of threads the differences between policies become more prominent. We have investigated alternative cases starting with steady state temperatures of different benchmarks, different scheduling time slices, and longer simulation durations. The results are consistent with the set we present in this section.

6.1. Random Policy

This policy is closest to a scheduling policy that is oblivious of the temperature profile of the threads as well as the heating patterns of the processor. Thread assignment is based on fairness and implemented through random number generation. Figure 3 illustrates the temperatures for individual blocks in Kelvins, where the x-axis represents the temperature samples at every 100K cycles (the values can be translated to time by multiplying with 0.1msec). As mentioned earlier we assume steady state block temperatures of Mgrid, and display single run of SPEC benchmarks for clarity. It is interesting to note that with random benchmark selection performs fairly well, the critical block temperatures for FPU_REG, FPU_B1 and IDU are eventually reduced. However, the temperatures are still above the threshold for over 96% of the time.

Figure 3. Random policy for 10msec time slice

6.2. AvgTemp Policy

The simplest temperature-aware scheduling policy involves representing each thread with a single temperature value. In this case we represented threads with average temperature of the processor. The average values are calculated as a weighted average, where the block areas are taken into consideration. As SPEC2000 traces has limited thermal variation within the benchmark, the AvgTemp policy is expected to reduce the temperatures. However, Figure 4 shows that although for most benchmarks average temperature profile is a good indicator, there are others such as Lucas and Swim that have more thermal variation and cause increase in the temperature of the thermally critical blocks gradually.
6.3. MaxTemp Policy

MaxTemp policy aims to extract maximum performance from the processor; it schedules the most aggressive threads consecutively. The thermal behavior of SPEC with MaxTemp is displayed in Figure 5. We assume that Mgrid has been running for several time slices and its steady state temperatures have been reached for the initial temperatures. The high performance high temperature threads keep the temperatures above the threshold temperature 98% of the execution time. For longer execution periods the threshold is exceeded 42% of the execution time.

6.4. MinTemp Policy

MinTemp policy has two operation modes:
For on-chip temperatures above the safety thermal threshold our goal is to manage the temperature of the thermally critical blocks on the chip to avoid the performance degradation of the dynamic thermal management schemes. In this case MinTemp policy selects the thread that has the minimum temperature for the current cycle’s hottest block.

Otherwise if the temperatures are below the safety thermal threshold the schedulers goal is balance the threads in keeping the temperatures within the boundaries, yet execute the thermally challenging threads on the cooler chip as much as possible. Assuming that the packaging is designed so that a percentage of challenging benchmarks are left for dynamic thermal management. We give priority in servicing these challenging traces during the cooler operation mode.

Figure 6. MinTemp Policy 3-Round-Robin runs with 10msec time slices

Figure 6 shows the execution of MinTemp policy in consecutive round robin scheme. Notice that this figure has longer execution duration just to illustrate the long term performance of MinTemp. The cooling times although appear shorter than other traces, is comparable in 100 msec range. It is worth noting that after 130msec, which is around the RC time constant necessary for the initial cooling, MinTemp is effective at keeping the maximum temperature below thermal threshold. The zigzag patterns in the figure illustrates that MinTemp interleaves the thermally challenging threads whenever the on-chip temperatures are below safety threshold. Figure 7 shows the thermal behavior of MinTemp policy with the initial temperatures for Gcc, which has a moderate thermal profile. Similar observations are valid for this case and consecutive round-robin runs displayed in Figure 16, where maximum on-chip temperatures are below thermal threshold more than 99% of the execution time.
6.5. Fetch Throttling

Current microprocessor architectures have reportedly integrated a linear throttle mechanism, stopping the CPU instruction fetch mechanism for short periods of time and allowing it to cool. A prior-generation PowerPC processor also used such fetch-throttling to conserve power, when needed [2]. Other related techniques such as: Decode throttling varies the number of instructions that can be decoded per cycle [3]. We compared the temperature-aware operating system scheduling policies with fetch throttling.

For the sake of comparison we used the same thread execution sequence that was used by MinTemp and throttle fetch above the temperature threshold value of 358K. We selected a coarse-grained aggressive fetch throttling policy in order to affectively reduce the temperatures below the thermal threshold. Figure 8 illustrates the effect of fetch throttling after Mgrid’s steady state temperature has been reached. The initial benchmark sequence until Lucas has not executed due to coarse grained fetch throttling. After the sequence completes we continue with the initial benchmark sequence that was throttled (Ammp, Mgrid, Swim, Applu, Galgel, Equake, Sixtrack and Facerec). For this specific scenario, the IPC is 0.808 compared to 1.02 for the analogous case in MinTemp policy. The corresponding 20% IPC degradation is due to the halt during the cooling time as opposed to executing complementary benchmarks in MinTemp.

Notice that this degradation is for the minimum temperature execution stream that was previously selected by MinTemp. When the experiment is repeated with MaxTemp execution stream accompanied with fetch throttling the performance degradation is significantly higher. Although 20% degradation is the upper bound as it is for single run of SPEC we have observed around 15% performance degradation for longer runs of MinTemp. The performance degradation depends on the baseline scheduling policy as well as severity of thermal problems. In general for thermally challenging benchmark sets the degradation can be severe.
6.6. Effects of Varying Scheduling Time Slices

In some operating systems such as Linux, operating system time slice length can be adjusted. We experimented on various scheduling time slices from 10 - 50 msec. The conclusions for the thermal-aware scheduling policies still hold for different time slices. The only difference we observed was that the heating and cooling behavior is stronger for the longer scheduling slices, as the thermally critical benchmarks have more time to heat the processor blocks.

7. Conclusions and Future Work
Today’s microprocessor architectures present unique thermal challenges. We investigated the effectiveness of temperature-aware thread scheduling as a thermal management technique. Since system thread scheduling is already incorporated in the microarchitectures, thermal-aware scheduling can be performed with virtually no performance degradation. Most of the architectures in the market today employ packaging systems designed for less than worst-case temperature, with the assumption that threads with absolute worst-case thermal behavior are of limited number in the distribution and can be dealt dynamically with hardware-level reactive dynamic thermal management techniques.

We investigated the processor heating/cooling behavior and showed that the RC time constant that determines the heating/cooling speed is in the range of 100 msec for the baseline POWER4-like architecture. The range of time constants enables effectively utilizing the thread scheduling infrastructure to manage the on-chip heating slowly. We experimented on traces generated from SPEC2000 benchmark set where 12 of the 25 benchmarks have maximum block temperatures above 360ºK, which is higher than our temperature threshold of 358ºK. Having 40% of the benchmarks thermally critical: quite aggressive, compared the current reported packaging assumptions with only 20% benchmarks are above the packaging threshold values.

Even with a thermally challenging benchmark set, our MinTemp policy is effective in reducing the temperature from a high initial temperature such as Mgrid steady state temperature. MinTemp manages to keep the temperatures lower than the thermal threshold 99% of the execution time for cases with average initial temperature values (such as Gcc) over longer periods of execution. The reason MinTemp is effective in reducing the temperature is the diversity of hotspot blocks and the effectiveness of the policy in interleaving the challenging threads. Furthermore, MinTemp adapts to the processor heating behavior through its distinct operation modes: For thermally critical cycles it aims to reduce the maximum block temperature through scheduling threads with lower temperature profiles for the corresponding block. For the non-critical cycles, the goal is to execute the thermally critical traces as much as possible, while keeping the temperatures below at safety range.

Our experimental analysis indicates that scheduling selection can result in temperature differences around 5ºC on average between different benchmarks for hottest blocks. Fetch throttling is also effective at reducing the maximum temperatures below the thermal threshold. However, the corresponding performance degradation can be up to 20% for a single run of SPEC over the execution sequence identical to MinTemp policy. For longer durations the degradation can still be significant depending on the severity of heating problems. Random thread selection and AvgTemp policies are comparable effectiveness.

Figure 9 displays the percentage of cycles above the thresholds for the investigated policies. It is important to notice that for conservative thresholds such as 364ºK none of the policies provide sufficient thermal alleviation. Also, notice that fetch throttling and MinTemp are the most effective in temperature reduction. However, for fetch throttling we used the scheduling sequences generated by MinTemp. In the case that fetch throttling is coupled with MaxTemp or other policies, its effectiveness degrades dramatically. Furthermore, fetch throttling has significant performance degradation for lower thresholds. MinTemp displays the percentage of cycles above the thresholds for steady state behavior of the policy and shows that the temperatures are effectively reduced for thermal thresholds of 358ºK and above. For lower temperature thresholds, the number of cycles above threshold increase dramatically, which implies that other hardware based DTM techniques have to be activated for lower thresholds.

Our preliminary results in this study indicate that thermal-aware operating system task scheduling can be effectively used to manage the on-chip temperatures. Our current and future works include implementation of these policies in the operating system kernel on a multi-core processor. For CMP architectures utilizing system and software thermal management techniques along with hardware DTMs enables more efficient thermal management. The increased number and diversity of cores and threads enable CMP architectures good candidates for effective task scheduling policies.

References