Introdução à Pesquisa em Informática

Escrevendo a "Fundamentação Teórica e Trabalhos Relacionados" de um projeto de pesquisa

PUC Minas
Instituto de Ciências Exatas e Informática

Lesandro Ponciano

Departamento de Engenharia de Software e Sistemas de Informação (DES)

Revisão Bibliográfica

Ou, “Revisão da Literatura”

Arcabouço Conceitual

Limitações/Lacunas na literatura

Arcabouço Teórico

Perguntas ainda não respondidas

Revisão Bibliográfica

Fundamentação Teórica

- Arcabouço Conceitual
- Arcabouço Teórico

- Limitações/Lacunas na literatura
- Perguntas ainda não respondidas

Trabalhos Relacionados

Fundamentação Teórica

Apresente e analise ...

conceitos e

teorias

… que são fundamentais para se entender a pesquisa que você está propondo

1ª Dica de Ouro

Mantenha o foco!

- Tudo que é apresentado deve ser relevante à hipótese ou à questão de pesquisa
- Não apresente algo simplesmente por que você acha interessante
- Evite distrair ou confundir o leitor

Trabalhos Relacionados

Trabalhos no “estado da arte”

Para cada trabalho relacionado discuta

- O que o trabalho se propõe a responder?
- Quais métodos ele emprega para responder?
- Quais respostas/resultados ele apresenta?
- Qual a relação desse trabalho com a pesquisa que você está propondo?

2ª Dica de Ouro

Ao estabelecer a relação de um trabalho com a sua proposta, pense em o que o trabalho relacionado...

- faz e que será reusado
- mostra e que será comparado
- não analisa/mostra e que será analisado/mostrado/complementado

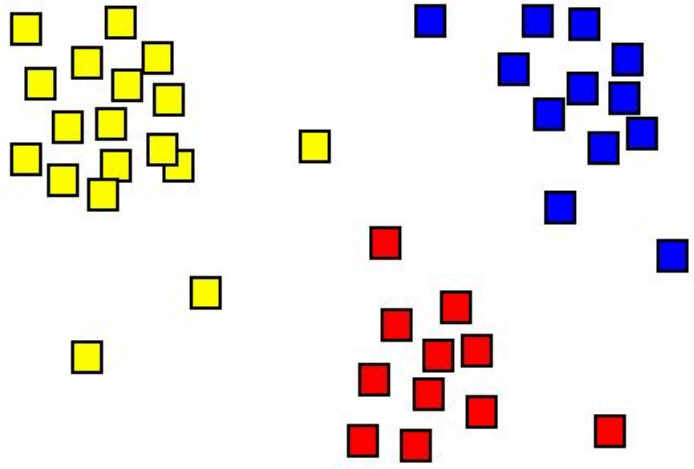
… na pesquisa que você está propondo

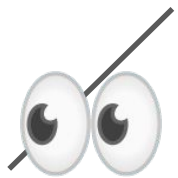
3ª Dica de Ouro

Organize e apresente os trabalhos relacionados de forma coerente

Exemplos

- Evolução histórica
- Abordagem metodológica
- Perspectiva teórica



Vamos analisar um exemplo? 

2012 Second International Conference on Cloud and Green Computing

Energy Efficient Computing through Productivity-Aware Frequency Scaling

Lesandro Ponciano, Andrey Brito, Lívia Sampaio, Francisco Brasileiro
Departamento de Sistemas e Computação
Universidade Federal de Campina Grande
Campina Grande, Brazil
Emails: {lesandrop, andrey, livia, fubica}@lsd.ufcg.edu.br

***Abstract*—This paper proposes a new policy for dynamic frequency scaling: productivity-aware frequency scaling (PAFS). PAFS aims at optimizing energy consumptions while still satisfying performance requirements of a given application. In contrast to the commonly-used ondemand frequency scaling, PAFS may keep the processor in a power save state even in high CPU-usage situations. This will be the case as long as the application (or set of applications) for which productivity is to be preserved presents acceptable performance (e.g., as stablished by a QoS contract). Our experiments show savings of up to 23.65% in energy consumption when compared to the commonly used ondemand DFS policy with no performance degradation for the productivity metric. PAFS is, therefore, binded to a single or a set of applications running in a machine. Nevertheless, compared to previous approaches to application-specific frequency scaling, PAFS does not require modifying the application or a calibration process. PAFS requires only a productivity metric which may already be exported by an application (e.g., through a log file, such as response time or throughput in an Apache webserver) or which may be computed through a simple program or script.**

***Keywords*-power-efficient computing, green IT, quality of service, frequency scaling**

decide in which frequency the processor should operate at each time. Usually, frequency scaling is performed by policies at compiler/application-level [4]–[6], operating system-level [7]–[10], or task-level [11]–[16].

Application-level policies define DFS at application design time. This approach allows optimizing both processor power consumption and application performance, but it requires the offline profilling and tuning of the application. System-level policies, in turn, do not consider application characteristics, and they change the processor frequency in response to variations on the system load. However, not knowing applications characteristics, system-level DFS will never achieve savings comparable to the ones achieveable with application-level tunning.

Finally, task-level policies are between these two approaches. This type of policy carries outs DFS taking into account some performance indicator (*e.g.*, service-level agreements – SLA, and application service level objectives – SLO). The present work focuses on task level DFS. In contrast to existing task-level approaches, ours generalizes

Ponciano, L., Brito, A., Sampaio, L., & Brasileiro, F. (2012). Energy Efficient Computing through Productivity-Aware Frequency Scaling. In 2012 Second International Conference on Cloud and Green Computing (pp. 191-198). IEEE. DOI https://doi.org/10.1109/CGC.2012.59

II. Background and Related Work

In this section, we provide some background information on how DFS is supported by modern hardware and discuss related work on frequency scaling techniques.

Como a seção de “Fundamentação Teórica”(*Background*) ficou curta, ela está concatenada com a seção de Trabalhos Relacionados (*Related Work*)

ergy savings and satisfying performance requirements without offline profiling and tuning, or assuming prior knowledge about application workloads;

- Our system enables binding savings and performance guarantees (*e.g.*, quality of service – QoS – guarantees) to keep the system in the most energy-efficient mode while meeting required performance levels (independently of the actual CPU load levels), but still is able to quickly react as situations change.
- We show that by achieving energy savings and productivity requirements, PAFS complements the range of options for frequency scaling policies: ondemand, performance, and powersave.

In the remainder of this paper, before detailing the experimental setup and presenting our results (Section IV), we review the relevant background and related work (Section II), and present our PAFS policy (Section III).

II. Background and Related Work

In this section, we provide some background information on how DFS is supported by modern hardware and discuss related work on frequency scaling techniques.

A. Background

Modern processors can run at a range of clock frequencies, for example, using Intel's SpeedStep [18] and AMD's Cool 'n' Quiet [19] technologies. Dynamic frequency scaling is a mechanism that allows scaling processor frequency by software instructions. This mechanism enables the reduction on power consumption by lowering the processor frequency (with a potential negative impact on system's performance). The power consumption in a processor is a nonlinear function of the operating frequency and voltage. Nevertheless, voltage and frequency are related; it is not possible to put the processor in a high frequency state without also increasing its supply voltage. Therefore, when frequency is changed, voltage is automatically changed to match requirements.

Adjusting the frequency (and power management in general) can be made through a platform-independent interface named Advanced Configuration and Power Interface (ACPI) [20]. Regarding frequency scaling, ACPI defines power-performance states (P-states). These states vary between P_0, the highest-performance state, and P_n the lowest-performance state.

B. Related Work

Processor frequency scaling has been studied from different perspectives and with different goals. According to the level at which frequency scaling is applied and whether performance metrics are considered or not, the studies may be broadly divided into three categories: application/compiler-level [4]–[6], task-level [11]–[16], and system-level [7]–[10].

Application/compiler-level policies perform DFS at application-level or with some compiler support, focusing on a specific infrastructure, performance metrics, and/or energy saving goals. For example, the Intel Energy Checker SDK [4] is an API to help constructing green software by exporting application progress metrics and importing energy consumption measurements (from hardware meters). Thus, the energy-efficiency policies can be implemented in the application using consumption measurements imported from the meters. The advantage of this approach is that frequency scaling decisions are aware of application current and future behavior, such as loops, recursive calls, message exchanges, and deadlines, allowing a more accurate DFS. On the other hand, the developer must be aware of energy consumption at the application design [4]–[6], or use a specific compiler to help on this task [21]. In other words, in design time both performance and energy need to be considered. In our approach, energy considerations are automated in run time.

The task-level category, in turn, comprehends policies that are not coupled to the application code, but are aware of running applications. Works in this category perform DFS taking into account some performance indicator (*e.g.*, application deadlines [13], [14], service-level agreements – SLA [12], [15], and application service level objectives – SLO [16]) or prior knowledge about the characteristics of the infrastructure workload (*e.g.*, resource utilization [11]). In general, the two main differences between our DFS approach and related works are that (*i*) we generalize the concept of performance in a productivity metric; and (*ii*) we do not require any prior information on system's behavior. Furthermore, our approach allows using the same policy with different productivity metrics, which are defined according to the user requirements.

System-level strategies perform DFS at the operating system without considering any application characteristics. A number of system-level policies can be found in the literature [7]–[10]. Particularly, some of them are broadly used in today's production systems based on both Linux and Windows [7], [8], namely: *Performance*, *Powersave*, and *Ondemand*. The Performance policy focuses on maximizing the application performance by setting the CPU to run at the highest supported frequency. At the opposite side, the Powersave policy focuses on minimizing the power consumption by setting the CPU to run at the lowest supported frequency.

Lastly, the Ondemand policy adapts the CPU frequency to the current system load. According to this policy, the system load is checked periodically, and, when the load rises above a predefined threshold, the CPU is set to run at the next higher frequency. Otherwise, if the load falls below another threshold, the CPU is set to run at the next lower frequency. In this work we compare our PAFS policy with these policies.

A. Background

Modern processors can run at a range of clock frequencies, for example, using Intel's SpeedStep [18] and AMD's Cool 'n' Quiet [19] technologies. Dynamic frequency scaling is a mechanism that allows scaling processor frequency by software instructions. This mechanism enables the reduction on power consumption by lowering the processor frequency (with a potential negative impact on system's performance). The power consumption in a processor is a nonlinear function of the operating frequency and voltage. Nevertheless, voltage and frequency are related; it is not possible to put the processor in a high frequency state without also increasing its supply voltage. Therefore, when frequency is changed, voltage is automatically changed to match requirements.

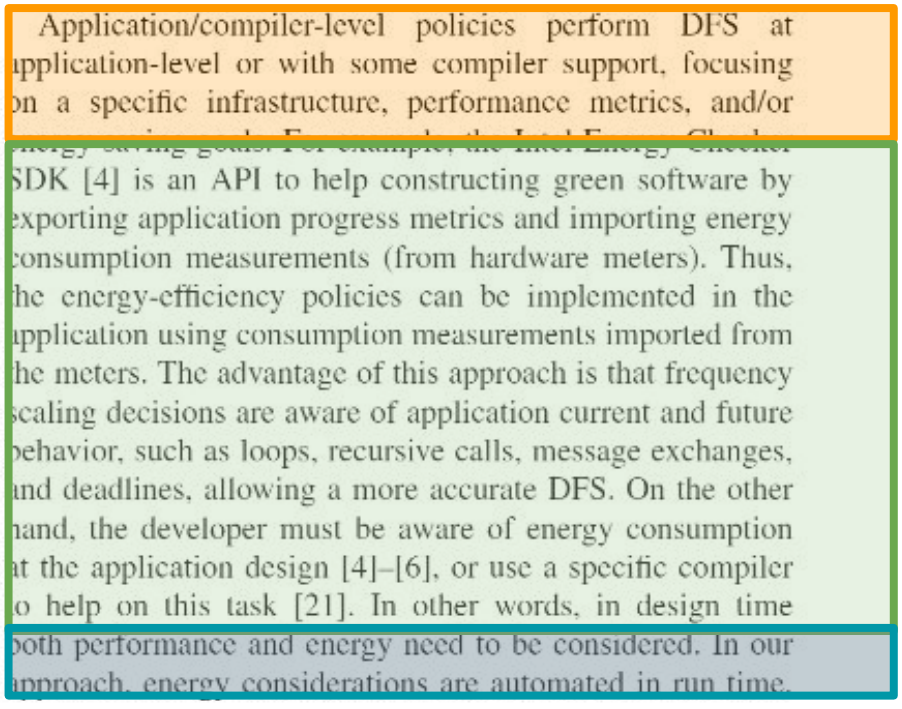
Adjusting the frequency (and power management in general) can be made through a platform-independent interface named Advanced Configuration and Power Interface (ACPI) [20]. Regarding frequency scaling, ACPI defines power-performance states (P-states). These states vary between P_0, the highest-performance state, and P_n, the lowest-performance state.

Apresenta os principais conceitos que são relevantes para que se possa compreender o que é feito/proposto no trabalho

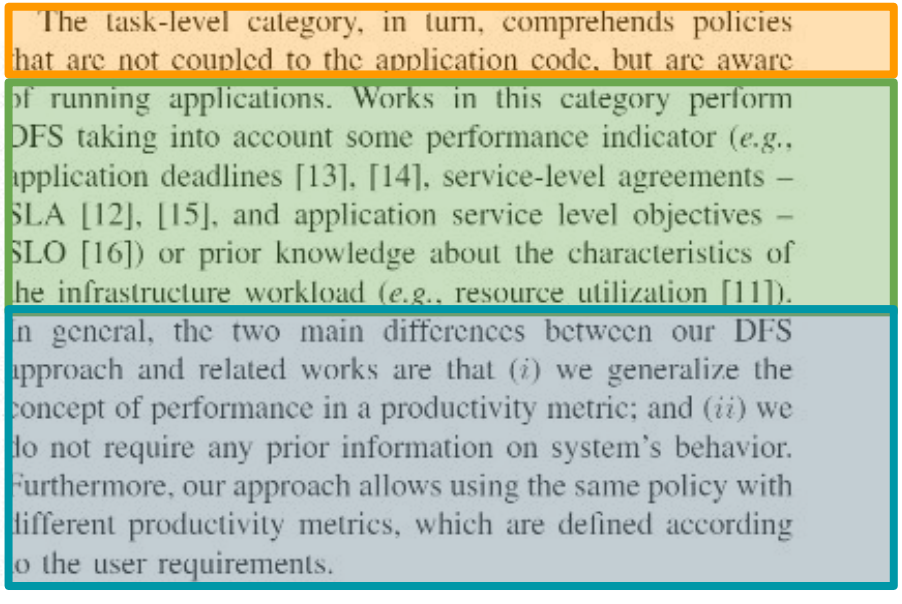
B. Related Work

Processor frequency scaling has been studied from different perspectives and with different goals. According to the level at which frequency scaling is applied and whether performance metrics are considered or not, the studies may be broadly divided into three categories: application/compiler-level [4]–[6], task-level [11]–[16], and system-level [7]–[10].

Agrupa os trabalhos relacionados em 3 categorias e nomeia as categorias


Application/compiler-level policies perform DFS at application-level or with some compiler support, focusing on a specific infrastructure, performance metrics, and/or

SDK [4] is an API to help constructing green software by exporting application progress metrics and importing energy consumption measurements (from hardware meters). Thus, the energy-efficiency policies can be implemented in the application using consumption measurements imported from the meters. The advantage of this approach is that frequency scaling decisions are aware of application current and future behavior, such as loops, recursive calls, message exchanges, and deadlines, allowing a more accurate DFS. On the other hand, the developer must be aware of energy consumption at the application design [4]–[6], or use a specific compiler to help on this task [21]. In other words, in design time both performance and energy need to be considered. In our approach, energy considerations are automated in run time.


The task-level category, in turn, comprehends policies that are not coupled to the application code, but are aware of running applications. Works in this category perform DFS taking into account some performance indicator (*e.g.*, application deadlines [13], [14], service-level agreements – SLA [12], [15], and application service level objectives – SLO [16]) or prior knowledge about the characteristics of the infrastructure workload (*e.g.*, resource utilization [11]).

In general, the two main differences between our DFS approach and related works are that (*i*) we generalize the concept of performance in a productivity metric; and (*ii*) we do not require any prior information on system's behavior. Furthermore, our approach allows using the same policy with different productivity metrics, which are defined according to the user requirements.

Em cada parágrafo

- apresenta a categoria de trabalhos relacionados
- detalha os trabalhos na categoria
- estabelece a relação com a pesquisa proposta

A imagem mostra apenas os 2 primeiros parágrafos da seção de Related Work

4ª Dica de Ouro

Sempre que ler um artigo científico, analise o estilo de escrita dele!

Isso ajudará você a

- julgar melhor se o artigo está bem escrito ou não
- aprender com o estilo empregado nos artigos bem escritos

Obrigado!

Bom Trabalho :)