[Comentário geral:

Nota do ar1: 8,0

A justificativa para a nota está espalhada no decorrer do texto com todas as minhas anotações.

Obs.: seu artigo não está no formato da SBC. Vc tem que pegar o template em LaTeX e reescrever o artigo naquele formato. Deve ter um link para o template na página do SBRC

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

Submissão ao SBRC:

Tenho dúvidas sobre a aceitação do comitê de programa mas acho que vale a pena tentar. Meu medo é de acharem que o artigo é muito mais sobre o desenvolvimento de uma ferramenta do que uma pesquisa completa. De qq forma acho que temos que tentar. Se não for aceito podemos resubmeter para o workshop de nuvens, grades e aplicações — o wcga, que tem prazo de submissão depois que sai o resultado do SBRC]

Program Autotuning with Cloud Computing and OpenTuner -> [Eu preferia que o título tivesse algo como "A new" ou "A ..." em que o ... fosse algum adjetivo/qualidade do mecanismo que vc bolou em relação ao que existe]

The sequential and empirical search takes even longer to nd optimizations for programs with considerable execution times. -> [Não entendi. Acho que o "longer" que me deixou perdido. nas frases anteriores vc não falou nada sobre algo demorar. Daí vem essa frase dizendo que demora mais. Fiquei perdido.]

extension -> [Do jeito que está vc vai apresentar um código que foi implementado sobre o OpenTuner. Acho que assim a contribuição fica reduzida a um código e vai ser difícil ser aceito. Do meu ponto de vista vc propõe mais do que um código pois pelo que eu lembre, essa ideia de usar a nuvem e paralelizar a execução de autotuners é sua não? Então vc tem que dizer que a contribuição é uma nova metodologia para execução de autotuners tirando proveito das redes de computadores e da computação em nuvem. A extensão é uma aplicação da sua metodologia, ou um caso de uso. A ideia é mostrar que a metodologia independe do autotuner usado. Vc tem que mexer no artigo para passar essa ideia]

approach -> approach. Results from experiments show that ...

[Tem que resumir os resultados já aqui no resumo do artigo]

Program Autotuning with Cloud Computing and OpenTuner

Pedro Bruel phrb@ime.usp.br
Alfredo Goldman gold@ime.usp.br
Daniel Batista batista@ime.usp.br

Instituto de Matemática e Estatística (IME)
Universidade de São Paulo (USP)
R. do Matão, 1010 – Vila Universitária, São Paulo – SP, 05508-090

Abstract. *The OpenTuner framework provides domain-agnostic tools for the implementation of autotuners. The framework sequentially evaluates program configurations exploring search spaces that are commonly very large. The sequential and empirical search takes even longer to find optimizations for programs with considerable execution times.*
This paper presents an extension to the OpenTuner measurement driver that distributes and parallelizes the measurement process using cloud computing resources from the Google Compute Engine (GCE). We compare the performance of our implementation to the unmodified framework using a diverse benchmark. We offer insight into the problem domains that benefit from this approach.

Pedro: A very short summary of the results will be provided in the abstract, as well as a brief discussion of the result normalization research question.

1. Introduction

Disclaimer: This is a draft of the paper. The results and implementations are not final or completed. Future versions will improve the presentation and present results and conclusions. The maximum page count for SBRC is 14 pages, and this document is 10 pages long.

The program autotuning problem fits in the framework of the Algorithm Selection Problem, introduced by Rice in 1976 [1]. The objective of an autotuner is to select the best algorithm, or algorithm configuration, for each instance of a problem. Algorithms or configurations are selected according to performance metrics such as the time to solve the problem instance, the accuracy of the solution and the energy consumed. The set of all possible algorithms and configurations that solve a problem defines a *search space*. Various optimization techniques search this space, guided by the performance metrics, for the algorithm or configuration that best solve the problem.

Pedro: I removed the Algorithm Selection Problem figure. It is marginally relevant for the paper's subject, but not needed to understand the contributions and results. Should I re-include it?

Should I re-include it? -> [No]

Autotuners can specialize in domains such as matrix multiplication [2], dense [3] or sparse [4] matrix linear algebra, and parallel programming [5]. Other autotuning frameworks provide more general tools for the representation and search of program configurations, enabling the implementation of autotuners for different problem domains [6, 7].

The OpenTuner framework [6] provides tools for the implementation of autotuners for various problem domains. It implements different search techniques that explore the same search space for program optimizations. Although support for parallel compilation is provided in the framework, the empirical exploration of the search space, that is, running and measuring program execution time, is done sequentially.

the -> , the

The main contribution of this paper is the implementation of an OpenTuner extension that distributes and parallelizes the exploration of the optimization space by combining results obtained from virtual machines ~~in a cloud computing environment~~. A local machine (LM) runs the main OpenTuner application and several virtual machines (VM) run measurement modules that provide results when requested, performing a more efficient exploration of the search space.

The main contribution of this paper is the implementation of an OpenTuner exten- sion -> [Não é. Vc tem que mudar essa frase para algo como: In this paper we propose a new methodology to run autotuners taking advantages of computer networks and cloud computing. The ideia is to … . To evaluate the advantages of the proposal, it is implemented as a OpenTuner extension…" e aí continua o parágrafo]

The interactions between the local and virtual machines follows the client-server model. The local machine runs a measurement client that requests results from various measurement servers running in virtual machines hosted at the Google Compute Engine (GCE). We compare the performance of our extension with the unmodified framework in a diverse benchmark of applications, identifying the problem domains that benefit from this cloud-based approach.

hosted at the Google Compute Engine (GCE) -> . In our experiments the virtual machines are hosted at the Google Compute Engine (CGE) but it is not restricted to this cloud. The only request is that the cloud provide an API … [explicar qual o requisito]

Pedro: Added hooks for the initials GCE, LM and VM.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 discusses the architecture of the OpenTuner framework. Section 4 presents the architecture of the measurement driver extension, the GCE interface and the application protocol ~~that mediates the interactions between *MeasurementClient* and *MeasurementServers*~~. Section 5 discusses the result normalization strategies. Section 6 describes the experiments performed and the applications used in the benchmark. Section 7 discusses the results. Section 8 concludes.

Section 4 presents the archi- tecture of the measurement driver extension -> Section 4 presents the methodology proposed and the archi- tecture of the measurement driver extension

2. Related Work

Rice's conceptual framework [1] formed the foundation of autotuners in various problem domains. In 1997, the PHiPAC system [2] used code generators and search scripts to automatically generate high performance code for matrix multiplication. Since then, systems tackled different domains with a diversity of strategies. Whaley *et al.* [3] introduced the ATLAS project, that optimizes dense matrix multiply routines. The OSKI [4] library provides automatically tuned kernels for sparse matrices. The FFTW [8] library provides tuned C subroutines for computing the Discrete Fourier Transform. In an effort to provide a common representation of multiple parallel programming models, the INSIEME compiler project [5] implements abstractions for OpenMP, MPI and OpenCL, and generates optimized parallel code for heterogeneous multi-core architectures.

multiply -> multiplication

Some autotuning systems provide generic tools that enable the implementation of autotuners in various domains. PetaBricks [9] is a language, compiler and autotuner that introduces abstractions, such as the `either...or` construct, that enable programmers to define multiple algorithms for the same problem. The ParamILS framework [7] applies stochastic local search methods for algorithm configuration and parameter tuning. The OpenTuner framework [6] provides ensembles of techniques that search spaces of program configurations. Bosboom *et al.* and Eliahu use OpenTuner to implement a domain specific language for data-flow programming [10] and a framework for recursive parallel algorithm optimization [11].

In a progression of papers [12, 13, 14], Gupta *et al.* provide experimental evaluations of the application of cloud computing to high performance computing, describing which kind of applications has the greatest potential to benefit from cloud computing. Their work highlights small and medium scale projects as the main beneficiaries of cloud computing resources.

Pedro: Provide a better discussion of Gupta et al.'s results.

Pedro: Justify the choice of OpenTuner as the modified system, by saying it is a domain-agnostic tool that has, to the best of our knowledge, no equivalent in scope.

[Aqui no final da seção faltou um último parágrafo que fale algo como:

For the best of our knowledge, there is no proposal in the literature that present a methodology to use cloud computing aiming the execution of auto tuners. By doing that it would be expected that ... . The proposal in Section 4 presents such a methodology and the results in Section 5 shows its advantages in relation to the conventional executions of

3. OpenTuner

[Como agora a principal contribuição seria a metodologia eu acho que é melhor vc mover essa seção para ser a 4 e trazer a 4 pra cá, isso considerando que o que eu propus — de dizer que tudo independe do opentuner — seja possível. Nesse caso ficaria esquisito ter uma seção só

OpenTuner search spaces are defined by *Configuration*s, that are composed of *Parameter*s of various types. Each type has restricted bounds and manipulation functions that enable the exploration of the search space. OpenTuner implements ensembles of optimization techniques that perform well in different problem domains. The framework uses *meta-techniques* to coordinate the distribution of resources between techniques. Results found during search are shared through a database. An OpenTuner application can implement its own search techniques and meta-techniques, making the ensemble more robust. The source code is available[1] under the MIT License.

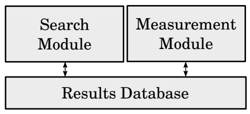

Figure 1. Simplified OpenTuner Architecture.

Figure 1 shows a high-level view of OpenTuner's architecture. Measurement and searching are done in separate modules, whose main classes are called *drivers*. The search driver requests measurements by registering configurations to the database. The measurement driver reads those configurations and writes back the desired results. Currently, the measurements are performed sequentially.

OpenTuner implements optimization techniques such as the Nelder-Mead [15] simplex method and Simulated Annealing [16]. A resource sharing mechanism, called *meta-technique*, aims to take advantage of the strengths of each technique by balancing the exploitation of a technique that has produced good results in the past and the exploration of unused and possibly better ones.

Pedro: Compare sequential and parallel tuning. Describe when OpenTuner uses threads to compile target programs.

Measurement Server and Client

Measurement Server and Client -> [Mudar isso aqui para falar da metodologia e só depois focar em código]

The extension follows the client-server model, distributing measurements of program configurations between a group of virtual machines running *MeasurementServer*s in the GCE. The servers waits for measurement requests from a client, and maintain copies of the program to be autotuned and the user-defined function that measures configurations.

The machine running the OpenTuner autotuner runs a *MeasurementClient*, an extension of the native *MeasurementDriver*, that instead of compiling and running result requests locally, uses the GCE interface to route requests to virtual machines and them saves the results to the local database.

[1] Hosted at GitHub: `github.com/jansel/opentuner`

[Em ambas as figuras eu senti falta de alguma delimitação mostrando o que está na nuvem e o que está fora. Poderia inclusive delimitar isso com a figura de uma nuvem de modo que dentro dela estejam os “módulos” que vão executar na nuvem]

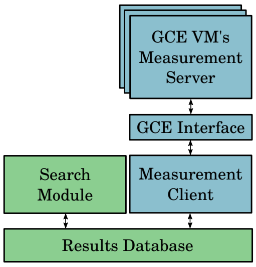

Figure 2. A high-level view of the architecture.

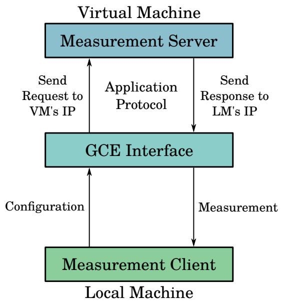

Figure 3. A lower-level view of the architecture.

Figures 2 and 3 present an overview of the architecture of the extension. Figure 2 presents the architecture of an OpenTuner application running the measurement client and communicating with the measurement servers. Green boxes in the figure represent OpenTuner modules that will not be modified, and blue boxes represent new or modified modules.

Figure 3 shows, on a lower level of abstraction, the interactions between the measurement client and servers. The client requests results from the server through a wrapper of the GCE Python API. The GCE interface also encapsulates the application protocol used in the client-server communication.

The remaining of this section describes the extension implementation in further detail, the GCE interface and the application protocol.

4.1. Implementation Details

OpenTuner controls the execution flow of an application with the `main` function of the `TuningRunMain` class. This function sets up the database and the search and measurement modules. It then calls the `main` function of the search driver, which runs the main loop of the application. The search driver generates configurations to be tested and saves them to the database. It then calls the `process_all` function of the measurement driver and blocks until the function returns.

Pedro: Listing 1 will be replaced by the complete implementation if it is short enough. Otherwise it will be removed.

The `process_all` function is able to compile programs in parallel, but the measure-

```python
class MeasurementClient(MeasurementDriver):
    """
    Reads DesiredResults and requests VMs
    in the cloud to compute Results.
    """
    def __init__(self,
            measurement_interface,
            input_manager,
            **kwargs):
        super(MeasurementDriver, self).__init__(**kwargs)
        """
        Instantiate and configure the VM Measurement
        Servers using the GCE interface.
        """

    def process_all(self):
        """
        Process all results in the
        database, by sending requests to
        VMs in the cloud and waiting
        for responses.
        """
```

Listing 1: Proposed modifications to `MeasurementDriver`.

ments are done sequentially. Listing 1 shows the functions of the measurement driver that were modified to enable the `MeasurementClient` to process results with Google Compute Engine resources.

During initialization the measurement client uses the GCE interface to start and configure virtual machines. The interface stores each measurement server's IP. When the search driver makes requests for results, the `process_all` function will route them to the servers via the GCE interface. The interface will call the appropriate GCE Python API functions, and wait for the responses.

Pedro: The description of the measurement server's internal will be added, detailing how the user program is copied to the virtual machines.

4.2. GCE Interface

Pedro: This section will describe how the GCE interface mediates the result requests and configuration measurements, how it initializes the virtual machines and obtains the user code.

The interactions between the local *MeasurementClient* and the virtual machines' *MeasurementServer*s are mediated by a wrapper of the GCE Python API, described in Section 4.2.

The utility functions and the measurement server's and client's code are available[2] under the GNU General Public License.

[2] All code is hosted at GitHub:

by a wrapper of the GCE Python API, -> [Detalhe demais de implementação. É desnecessário]

License -> License at ... [colocar o link do github]

4.3. Application Protocol

Pedro: The application protocol will be described here. The messages exchanged by the server and client and their effects will be specified.

5. Result Normalization

Pedro: This section will discuss the different strategies in depth, and relate them to the experiments performed.

Using a cloud environment, an autotuner will typically optimize programs for a machine with a different architecture from the virtual machines. A normalization technique must be devised that enables the results found in the virtual machines to be valid for the local machine. We present four approaches to this problem. The best approach for each problem domain must be experimentally determined, and could be a combination of the approaches described here.

machines -> machines. For example … [dar um exemplo claro para o leitor entender o que vc quer dizer. Falar que se vc quer fazer autotuning numa máquina x mas a VM está numa máquina com arquitetura y, …]

Autotune Performance Models Another autotuner could be implemented to optimize parameters of a simple performance model, that would associate a configuration's measurement and the virtual machine that produced it with a conversion function that transposes performance results to the target architecture.

Ensembles of Virtual Machines The cloud application could be composed of virtual machines with different architectures. The final performance measurement for a configuration would be built from some combination of the results obtained in these different virtual machines.

Architecture Simulators The target machine could be modeled by an architecture simulator such as *zsim* [17], a simulator for multi-core architectures available[3] under the GNU General Public License. Using a simulator would solve the normalization problem but introduce other problems, such as the simulator's accuracy and performance.

zsim [17] -> [Não há uma referência de artigo sobre esse zsim? Se não tem, ok usar a URL mas ponha a url no bibtex e referencia ela com \cite. Não ponha nota de rodapé. E sempre que for referenciar urls, no bibtex junto precisa ter a data de último acesso]

`github.com/phrb/measurement-server`
`github.com/phrb/autotuning-gce`

[3] Hosted at GitHub: `https://github.com/s5z/zsim`

Autotune in the Cloud Finally, the normalization problem could be sidestepped, at least in initial stages of research, by running the servers and clients in the cloud using the same kind of virtual machine.

6. Experiments

Pedro: The experimental settings will be described in this section.

7. Results

Pedro: This section will present and discuss the results, connecting the findings with normalization techniques and problem domains.

8. Conclusion

This paper presented an extension of the OpenTuner autotuning framework enabling it to leverage the cloud computing resources from GCE. We propose four approaches to solve the result normalization problem which would enable transposing the results obtained in virtual machines to a local machine.

References

[1] J. R. Rice, "The algorithm selection problem," 1976.

[2] J. Bilmes, K. Asanovic, C.-W. Chin, and J. Demmel, "Optimizing matrix multiply using phipac: A portable, high-performance, ansi c coding methodology," in *Proceedings of the 11th International Conference on Supercomputing*, ser. ICS '97. New York, NY, USA: ACM, 1997, pp. 340–347.

[3] R. C. Whaley and J. J. Dongarra, "Automatically tuned linear algebra software," in *Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*, ser. SC '98. Washington, DC, USA: IEEE Computer Society, 1998, pp. 1–27.

[4] R. Vuduc, J. W. Demmel, and K. A. Yelick, "Oski: A library of automatically tuned sparse matrix kernels," in *Journal of Physics: Conference Series*, vol. 16, no. 1. IOP Publishing, 2005, p. 521.

[5] H. Jordan, P. Thoman, J. J. Durillo, S. Pellegrini, P. Gschwandtner, T. Fahringer, and H. Moritsch, "A multi-objective auto-tuning framework for parallel codes," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*. IEEE, 2012, pp. 1–12.

This paper presented an extension of the OpenTuner autotuning framework enabling it to leverage the cloud computing resources from GCE. -> [Modificar levando em conta tudo que eu falei sobre a implementação ser importante mas não ser a principal contribuição]

We propose four approaches to solve the result normalization problem which would enable transposing the results obtained in virtual machines to a local machine. -> [Essas 4 abordagens tb devem ser conectadas no início do texto como contribuição. Então vc precisa mexer no artigo para deixar claro que as contribuições principais são:

- uma metodologia para executar autotuners de modo a tirar proveito do poder computacional e do paralelismo das nuvens
- 4 abordagens para realizar a normalização decorrente das diferentes arquiteturas de máquinas na nuvem

contribuição secundária é:

- uma implementação da metodologia usando o OpenTuner e a CGE

]

[Tem muita referência. Pelo tamanho que um artigo do SBRC costuma ter, o ideal é que vc tenha umas 15 referências no máximo, que vai dar mais ou menos 1 página. Mais que isso vc vai ter problemas com o espaço]

"The algorithm selection problem," 1976. -> [É um livro? Está faltando mais dados. Se for um livro, qual a editora, se for um artigo, qual a conferência ou revista, qual o número de páginas dos anais?]

[6] J. Ansel, S. Kamil, K. Veeramachaneni, J. Ragan-Kelley, J. Bosboom, U.-M. O'Reilly, and S. Amarasinghe, "Opentuner: An extensible framework for program autotuning," in *Proceedings of the 23rd international conference on Parallel architectures and compilation*. ACM, 2014, pp. 303–316.

[7] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, "Paramils: an automatic algorithm configuration framework," *Journal of Artificial Intelligence Research*, vol. 36, no. 1, pp. 267–306, 2009.

[8] M. Frigo and S. G. Johnson, "Fftw: An adaptive software architecture for the fft," in *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, vol. 3. IEEE, 1998, pp. 1381–1384.

[9] J. Ansel, C. Chan, Y. L. Wong, M. Olszewski, Q. Zhao, A. Edelman, and S. Amarasinghe, "Petabricks: A language and compiler for algorithmic choice," in *ACM SIGPLAN Conference on Programming Language Design and Implementation*, Dublin, Ireland, Jun 2009.

[10] J. Bosboom, S. Rajadurai, W.-F. Wong, and S. Amarasinghe, "Streamjit: a commensal compiler for high-performance stream programming," in *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*. ACM, 2014, pp. 177–195.

[11] D. Eliahu, O. Spillinger, A. Fox, and J. Demmel, "Frpa: A framework for recursive parallel algorithms," Master's thesis, EECS Department, University of California, Berkeley, May 2015.

[12] A. Gupta, L. V. Kalé, D. S. Milojicic, P. Faraboschi, R. Kaufmann, V. March, F. Gioachin, C. H. Suen, and B.-S. Lee, "Exploring the performance and mapping of hpc applications to platforms in the cloud," in *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*. ACM, 2012, pp. 121–122.

[13] A. Gupta, P. Faraboschi, F. Gioachin, L. V. Kale, R. Kaufmann, B.-S. Lee, V. March, D. Milojicic, and C. H. Suen, "Evaluating and improving the performance and scheduling of hpc applications in cloud," 2014.

[14] A. Gupta, L. V. Kale, F. Gioachin, V. March, C. H. Suen, B.-S. Lee, P. Faraboschi, R. Kaufmann, and D. Milojicic, "The who, what, why, and how of high performance computing in the cloud," in *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, vol. 1. IEEE, 2013, pp. 306–314.

[15] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The computer journal*, vol. 7, no. 4, pp. 308–313, 1965.

[16] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi *et al.*, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[17] D. Sanchez and C. Kozyrakis, "Zsim: fast and accurate microarchitectural simulation of thousand-core systems," in *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3. ACM, 2013, pp. 475–486.