Darwin: uma *framework* de otimização clusterizável

Henrique A. Rusa[1], **Kleber Kruger**[1], **Rodolfo Azevedo**[1]

[1]Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)
13.083-852 – Campinas – SP – Brasil

{h192050, k230226}@dac.unicamp.br

{rodolfo}@ic.unicamp.br

Abstract. *This article introduces Darwin, a meta-heuristics based optimization tool. It has implemented various optimization algorithms: genetic algorithm, particle swarm and differential evolution. It also has two backends for execution, the first one focused on the use of a cluster and the second one on the use of local computational resources. The x86 Sniper architecture simulator was used together with the Parsec benchmark, optimizing cache parameters, to validate the developed tool. At last, it is evaluated that the tool allows the optimization using the algorithms in a simplified and parallelable way.*

Resumo. *Neste artigo apresentamos Darwin, uma ferramenta de otimização de aplicações a partir do uso de algoritmos meta-heurísticos. A ferramenta dispõe de três otimizações: algorítmo genético, particle swarm e evolução diferencial. Conta também com dois backends para execução, o primeiro focado no uso de um cluster e o segundo no uso de recursos computacionais locais. Utilizou-se o simulador de arquitetura x86 Sniper com o benchmark Parsec, otimizando-se parâmetros de cache, para validar a ferramenta desenvolvida. Ao final, avalia-se que a ferramenta permite a otimização utilizando somente algoritmos de forma simplificada e paralelizável.*

1. Introdução

Neste trabalho apresentamos a ferramenta *darwin* (`https://github.com/henriqar/darwin`) que tem por objetivo executar diferentes otimizações para problemas modelados por um conjunto finito de parâmetros e uma correspondente função de avaliação, ou seja, avaliados através de algoritmos meta-heurísticos. Baseado fortemente na biblioteca libOPT [Papa et al. 2017] a ferramenta expõe um meio no qual a otimização por meta-heurística seja possível através da implementação de uma interface utilizada para delegar automaticamente tarefas de simulação para *backends* de execução, como também pela descrição dos parâmetros da otimização para a ferramenta, que mapeia os possíveis conjuntos desses parâmetros para serem executados nos *backends* descritos. Como o *darwin* não possui conhecimento sobre a aplicação, a função de avaliação - ou função de *fitness* - implementa a avaliação dos pontos do espaço de pesquisa (definida pelo conhecimento do usuário), possibilitando a otimização do problema sem a necessidade do usuário recriar os algoritmos meta-heurísticos descritos neste trabalho.

2. Apresentação da Ferramenta

O algoritmo desenvolvido pode ser visto no diagrama da Figura 1. A lógica de execução baseia-se na avaliação de partículas (coordenadas do espaço de soluções), através de uma

função de *fitness* que qualifica cada resultado de acordo com um critério desejado. Um *batch* de partículas são executadas no *backend* escolhido e, a cada tempo de espera - ou *refresh_time* - o sistema é questionado sobre quais execuções encerraram. Quando todas as tarefas agendadas finalizarem a função de *fitness* é acionada para avaliar a qualidade dos resultados das partículas. Posteriormente o algoritmo de otimização é aplicado sobre as avaliações obtidas.

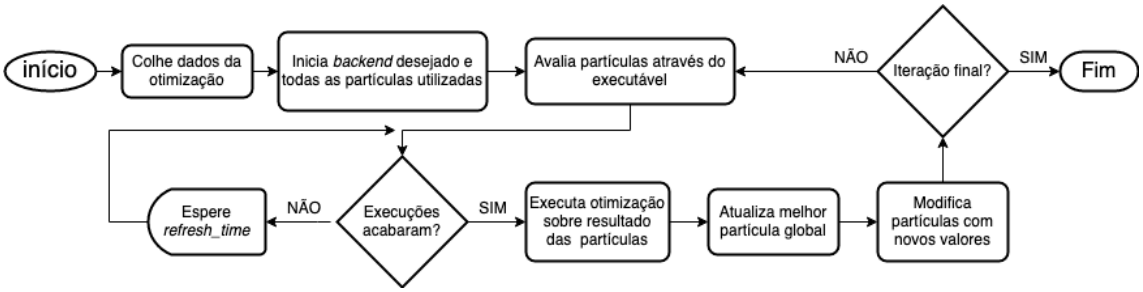

Figura 1. Algoritmo base da ferramenta *darwin*

Três algoritmos de otimização foram implementados: *Genetic Algorithm* [Goldberg 1989], o qual classifica as partículas e as seleciona para reprodução e disseminação de suas características; *Differential Evolution* [Storn and Price 1997], cuja formação de novas partículas é realizada por um cálculo diferencial dos parâmetros das partículas envolvidas; e o *Particle Swarm Optimization* [Eberhart et al. 2001] que se baseia em uma população de partículas que convergem para o melhor *fitness* global, idealmente. Além dos diferentes tipos de otimizações a aplicação desenvolvida provê dois *backend* de execuçã: HTCondor, o qual gerencia um *cluster* HTCondor [Thain et al. 2005] capaz de lidar com altas cargas de trabalho utilizando filas de execução para as simulações; e *Task Spooler* para execução dos trabalhos localmente (podendo ser paralelizável) através da aplicação *task spooler* [Rossell 2019].

3. Estudo de Caso

A fim de trabalhar com uma carga de trabalho representativa utilizou-se como base o simulador Sniper [Carlson et al. 2014] para simular o desempenho das *caches* de um processador com arquitetura Nehalem da Intel, alterando os parâmetros da mesma e verificando a combinação resultante com melhor desempenho para uma aplicação executada pelo processador. Também utilizou-se uma medida de potência para as *caches* utilizando o *software* McPAT [Li et al. 2009] integrado com o simulador Sniper. O *benchmark* Parsec [Bienia 2011], versão 2.1, com a aplicação *canneal* e tamanho de entrada *small* foi definido como carga de trabalho.

Os parâmetros escolhidos para serem otimizados foram a associatividade, tamanho e política de substituição das *caches* de instrução, dados e L2 do processador simulado com as seguintes escolhas: associatividades de 1, 2, 4, 8 e 16; política de substituição MRU, NRU, aleatória ou *Round Robin*; tamanhos de *cache* de instrução e dados com 8, 16, 32 ou 64 Kb; tamanho de *cache* L2 com 128, 256 ou 512 Kb. O algoritmo genético será utilizado na otimização dos parâmetros citados. O cluster foi utilizado com 14 partículas para exploração e 35 iterações. O *backend* local foi utilizado com 6 partículas, 15 iterações e 3 núcleos físicos de processamento paralelo.

3.1. Função de Avaliação

Definiu-se que as *caches* do processador serão otimizadas e, para isso, duas medidas foram fixadas para o cálculo do *fitness* das configurações encontradas: tempo de execução (em segundos) do *benchmark canneal* (foco da otimização) e potência dissipada (em Watts). A medida da potência é utilizada pois limita o aumento desproporcional do tamanho da *cache* devido à dissipação de potência pela área que a *cache* ocupa no silício. Adicionalmente, qualquer limitação na medida da potência pelo simulador (possíveis erros encontrados) foram considerados como *fitness* máximo.

A função de *fitness*, com as considerações discutidas, resulta em $fitness = time \cdot power$. A avaliação, portanto, suprimirá o crescimento ilimitado das *caches* do sistema pelo efeito negativo do aumento do tamanho - dissipação elevada de potência. Note que a equação de *fitness* obtém a energia despendida pelas estruturas ($J = W \cdot s$).

3.2. Medidas Realizadas

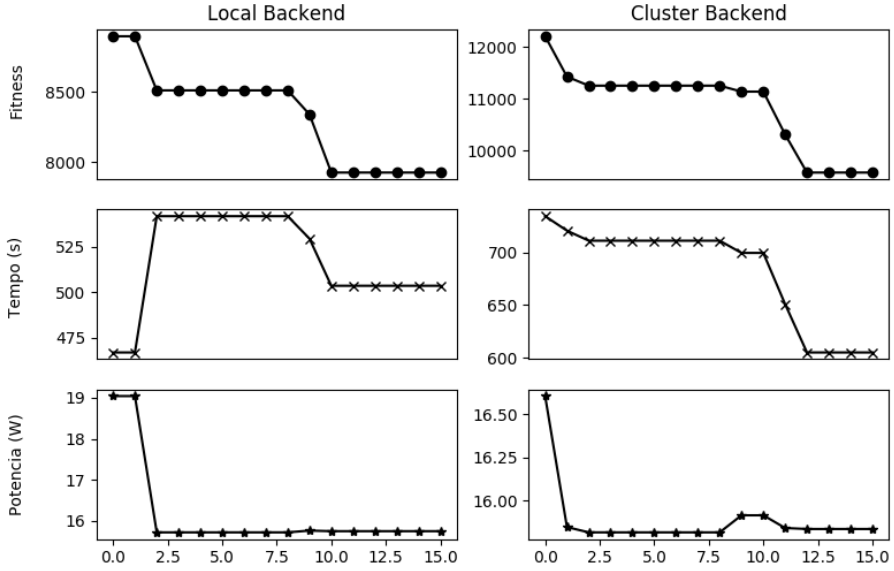

Figura 2. Evolução das partículas com relação à medida de *fitness*, tempo de execução (s) e potência dissipada (W) para os *backends* disponíveis.

A figura 2 apresenta os resultados para as otimizações de *cache* com os dois *backends* da ferramenta. Para o *backend* local é fácil ver que inicialmente a potência cai aliado a um aumento do tempo de execução do *benchmark*; nas iterações seguintes a ferramenta otimiza os tempos de execução mantendo a potência dissipada. A configuração com melhor resultado para o *backend* local obteve associatividade 1 para as *caches* de instrução, dados e L2, política de substituição MRU para as *caches* de instrução, dados e L2, 8 Kb de *cache* de instrução e de dados e 128 Kb de *cache* L2. Para o *cluster* a otimização também melhorou inicialmente a potência dissipada e, consequentemente, reduziu o tempo de execução do *benchmark*. O *cluster* obteve associatividade 1 para as *caches* de instrução, dados e L2, política de substituição NRU para a *cache* de instrução,

política de substituição MRU para a *cache* de dados, política de substituição aleatória para a *cache* L2, 32 Kb de *cache* de instrução, 8 Kb de *cache* de dados e 128 Kb de *cache* L2.

Pode-se notar que existem poucas diferenças entre os resultados das otimizações, entretanto localmente obteve-se um melhor desempenho para o tempo de execução; já no *cluster* a dissipação de potência atingiu um melhor resultado. É interessante observar que mesmo com os recursos limitados o *backend* local conseguiu o melhor resultado de *fitness*, demonstrando que a ferramenta é de valia para qualquer usuário que deseje utilizar das técnicas implementadas mesmo sem o acesso a uma melhor infra-estrutura computacional.

4. Conclusão

A ferramenta *darwin* permite a qualquer usuário o acesso a uma gama de otimizações chamadas meta-heurísticas de forma simplificada e paralelizável, mesmo este provido dos recursos computacionais avançados como um *cluster* ou a partir da instalação de uma simples aplicação para gestão de processos. Os algoritmos implementados demonstram uma capacidade de auxiliar os pesquisadores de diversas áreas do conhecimento, abrindo novas oportunidades para a avaliação e análise de seus problemas com um sistema que é capaz de absorver o conhecimento através da especificação e desenvolvimento de um espaço de soluções significativo e uma função de avaliação objetiva e eficaz.

Referências

Bienia, C. (2011). *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University.

Carlson, T. E., Heirman, W., Eyerman, S., Hur, I., and Eeckhout, L. (2014). An evaluation of high-level mechanistic core models. *ACM Transactions on Architecture and Code Optimization (TACO)*.

Eberhart, R. C., Shi, Y., and Kennedy, J. (2001). *Swarm intelligence*. Elsevier.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition.

Li, S., Ahn, J. H., Strong, R. D., Brockman, J. B., Tullsen, D. M., and Jouppi, N. P. (2009). Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 469–480. ACM.

Papa, J. P., Rosa, G. H., Rodrigues, D., and Yang, X. (2017). Libopt: An open-source platform for fast prototyping soft optimization techniques. *CoRR*, abs/1704.05174.

Rossell, L. B. I. (2004–2019). ts - task spooler. a simple unix batch system. `https://launchpad.net/ubuntu/+source/task-spooler/0.7.5-1`.

Storn, R. and Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359.

Thain, D., Tannenbaum, T., and Livny, M. (2005). Distributed computing in practice: the condor experience. *Concurrency and computation: practice and experience*, 17(2-4):323–356.