Um Algoritmo Paralelo para Priorização de Testes Baseada em Similaridade usando OpenMPI

Carlos Diego Nascimento Damasceno[1], **Paulo S. L. Souza**[1], **Adenilso Simao**[1]

[1] Instituto de Ciências Matemáticas e de Computação – ICMC
Universidade de São Paulo – USP
Av. Trabalhador São-carlense, 400 – 13566-590 – São Carlos – SP – Brasil

damascenodiego@usp.br, {pssouza,adenilso}@icmc.usp.br

Resumo. *Priorização de testes baseada em similaridade é uma abordagem que usa funções de similaridade para priorizar casos de teste dos pares mais distintos para os mais semelhantes. Neste contexto, o cálculo de matrizes de similaridade desempenha um papel importante, porém caro ($O(n^2)$), pois essas matrizes são base para o processo de priorização. Algoritmos paralelos têm sido usados para otimizar o cálculo dessas matrizes no mapeamento de ontologias e modelagem de hardware, entretanto isso ainda não foi estudado no contexto de priorização de testes. Este artigo propõe um algoritmo paralelo de priorização de testes baseada em similaridade usando OpenMPI. Conjuntos de teste de diferentes tamanhos foram priorizados usando algoritmos paralelo e sequencial e comparados usando tempo de execução médio, speedup e eficiência.*

1. Introdução

Priorização de testes visa buscar uma ordem eficiente de execução para testes que maximize um dado critério, ainda que essa execução seja interrompida prematuramente [Yoo and Harman 2012]. Priorização baseado em similaridade é uma abordagem recente que usa funções de similaridade para guiar a priorização. Nela, assume-se que testes parecidos tendem a cobrir partes similares de um sistema e detectar defeitos semelhantes, logo não há ganho com sua execução simultânea [Cartaxo et al. 2011].

Para se realizar uma priorização baseada em similaridade, uma *matriz de similaridade* (SM) contendo o grau de similaridade entre todos os pares de teste deve ser calculada para um conjunto de n testes ao custo de $O(n^2)$. Após esse cálculo, a ordenação de testes pode ser efetuada usando um algoritmo de priorização usando distância máxima local (sigla do inglês LMDP [Henard et al. 2014]) que seleciona os pares de teste de menor similaridade, ordenando-os dos mais distintos aos mais semelhantes.

O cálculo de SM é também uma etapa do mapeamento de ontologias [Gîză-Belciug and Pentiuc 2015] e modelagem de hardware [Rawald et al. 2015]. Nesses domínios, algoritmos paralelos já foram estudados e forneceram melhorias significativas de performance. Entretanto, não foram encontrados estudos em priorização baseada em similaridade.

Nesse artigo propõe-se um algoritmo paralelo para calcular SM e uma variação paralela do LMDP usando OpenMPI [Open MPI 2016]. O tempo médio de execução, o *speedup*, e a eficiência desse algoritmo foram mensurados usando uma implementação sequencial e um conjuntos de teste com $50, 100, 200,$ e 300 casos de teste. Nesse estudo os

testes priorizados foram gerados usando teste baseado em modelos [Cartaxo et al. 2011]. Entretanto, nosso algoritmo pode ser usado em qualquer domínio que disponha de uma função de similaridade de testes.

2. Priorização paralela de testes baseada em similaridade

Na priorização baseada similaridade, o critério a ser maximizado descreve a similaridade calculada para os pares de casos de teste na forma de uma matriz SM de custo $O(n^2)$. De acordo com Zhang et al. [Zhang et al. 2017], o cálculo da matriz SM é um problema de comparação todos-contra-todos típicamente resolvido usando algoritmos paralelos do tipo *Mestre+Escravos* onde um processo atua como *mestre* gerenciando dados para processos *escravos* que performam computações parciais.

Nosso algoritmo paralelo de geração de matrizes de similaridade (sigla em inglês PGSM) usa np processos OpenMPI. Um processo *mestre* é responsável por distribuir os n casos de teste para $np - 1$ escravos de modo que eles calculem uma parte de SM. Cada escravo calcula uma sequência contígua de no máximo $\frac{nr(nr-1)}{2(np-1)}$ graus de similaridade de modo a garantir um balanceamento aproximado de tarefas. A matriz SM abaixo exemplifica a alocação da tarefa de cálculo graus de similaridade (d_s) entre três processos escravos ($p_{slv}, 1 \leq slv \leq 3$) para um total de quatro casos de teste (i.e. $np = 4$).

$$
\text{SM} = \begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \\ t_4 \end{array}
\begin{array}{c}
\begin{array}{cccc} t_1 & t_2 & t_3 & t_4 \end{array} \\
\left[\begin{array}{cccc}
0 & d_s(t_1,t_2) \in p_1 & d_s(t_1,t_3) \in p_1 & d_s(t_1,t_4) \in p_2 \\
0 & 0 & d_s(t_2,t_3) \in p_2 & d_s(t_2,t_4) \in p_3 \\
0 & 0 & 0 & d_s(t_3,t_4) \in p_3 \\
0 & 0 & 0 & 0
\end{array} \right]
\end{array}
$$

Essencialmente, o algoritmo PLMDP (LMDP paralelo), ordena casos de teste a partir dos pares mais distintos para os menos dissimilares usando uma matriz de similaridade particionada (SM_p) entre $np - 1$ escravos (Algoritmo 1).

Algoritmo 1 Parallel LMDP (PLMDP)

```
 1: IN: T = {t_1, t_2, ..., t_nr}, np − 1 escravos com seus SM_p / OUT: TCS      ▷ Testes priorizados
 2: TCS ← []
 3: enquanto #T > 0 faça
 4:     se #T > 1 então
 5:         AllReduce(d_s, p, MAXLOC)                                  ▷ max(d_s(t_i, t_j)) ∈ SM_p
 6:         Broadcast(t_i, t_j, p)                                     ▷ p broadcast t_i e t_j
 7:         TCS.add(t_i); TCS.add(t_j)
 8:         T ← T\{t_i, t_j}                                           ▷ mestre e escravos
 9:     senão TCS.add(t_i) where t_i ∈ T ;  T ← ∅
10:     fim se
11: fim enquanto
12: return TCS
```

A busca do par $\langle t_i, t_j \rangle$ mais distinto e seu processo detentor p é feita usando a rotina *AllReduce* com *MAXLOC* do OpenMPI (linha 5). Após isso, uma mensagem *broadcast* é transmitida para que as similaridades associadas ao par em questão sejam desconsideradas. Caso haja mais de um par com similaridade idêntica, o primeiro par é selecionado. Esse processo é repetido até que todos os testes sejam priorizados (TCS).

3. Resultados

Duas versões, uma paralela e outra serial, do algoritmo apresentado foram implementadas usando C++ and OpenMPI[1]. Elas foram avaliadas em um cluster de 8 nós Intel® Core™ i7-4790, 32Gb de RAM, 500Gb of HD, Ubuntu 14.04 64 bits, GCC 4.8.4, e OpenMPI 1.8.3 [Open MPI 2016]. Os algoritmos foram executados 20 vezes em conjuntos com $NR = \{50, 100, 200, 300\}$ casos de teste. Para analisar a escalabilidade do algoritmo proposto, nós também variamos o número de processos em $NP = \{2, 4, 6, 8\}$ com um processo para cada nó, totalizando 320 execuções. O speedup e a eficiência foram respectivamente calculados como $speedup = \frac{\Delta t_{serial}}{\Delta t_{parallel}}$ e $eff = \frac{\Delta t_{serial}}{\Delta t_{parallel} \times np}$ [Pacheco 2011].

A Figura 1 mostra o tempo médio do algoritmo PLMDP em função de NP e NR. Houve uma relação inversamente proporcional entre $avg(\Delta t)$ e NP. Para valores de NP maiores, cresce o número de cálculos de similaridade que podem ser efetuados simultaneamente. Dessa forma, esperou-se uma redução no tempo de execução e um incremento de speedup, como visto na Figura 2.

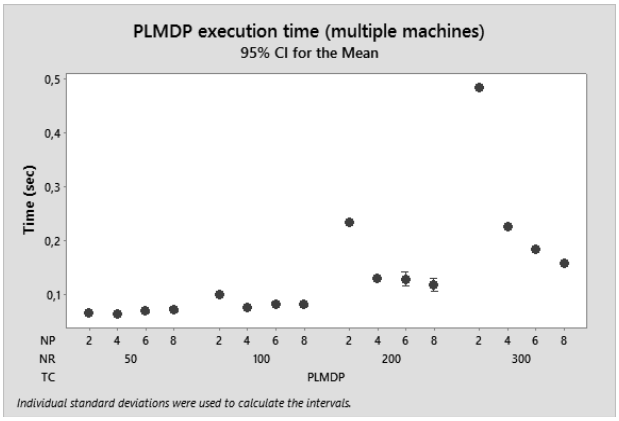

Figura 1. Tempo médio de execução do PLMDP

A análise do *speedup* mostrou um comportamento crescente em função de NP. Para $NR = 2$, o speedup decaiu conforme NR cresceu, gerando uma sobrecarrega. O crescimento do speedup começa a ocorrer para $NR \geq 2$ Ao analisar o speedup em função do tamanho das entradas, pode se notar que o incremento do speedup é bastante acentuado para $NR = \{200, 300\}$. Apesar de apresentar um bom speedup, o algoritmo PLMDP não se mostrou eficiente para valores crescentes de NP, como mostrado na Figura 3.

Essa queda na eficiência se justifica na forma como o cálculo dos graus de similaridade entre pares de testes foi distribuído no algoritmo PGSM. Durante a priorização efetuada pelo PLMDP, as consultas de *par de testes mais distinto* são prosseguidas da remoção de todos os graus de similaridade relacionados aos testes desse par. Após algumas iterações, esse passo pode levar um processo escravo a descartar todos os seus testes e graus de similaridade e entrar em estado ocioso, deixando de participar do processo de priorização, o que gera um *overhead*.

[1] `https://github.com/damascenodiego/fsmPrioritization`

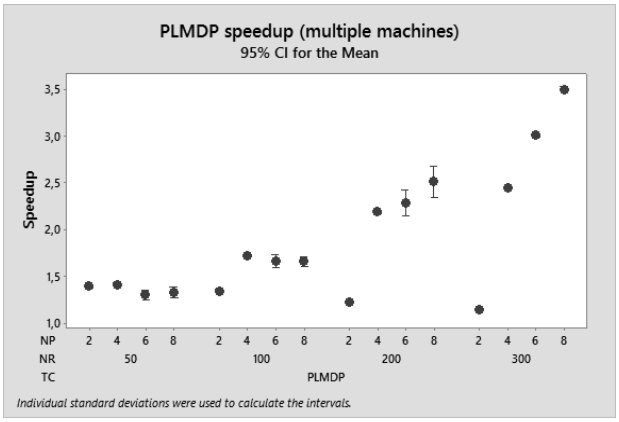

Figura 2. Speedup do PLMDP

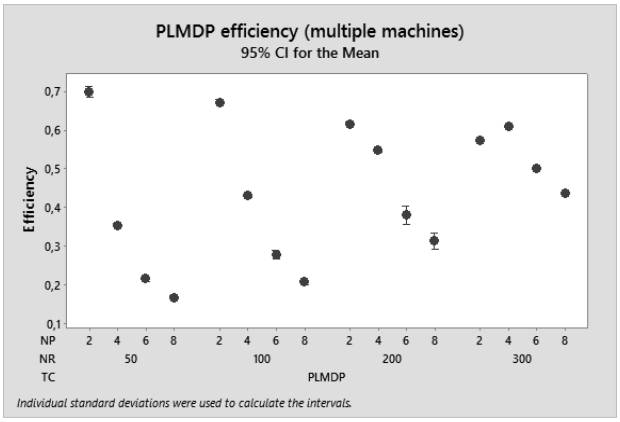

Figura 3. Eficiência do PLMDP

4. Considerações Finais

Priorização de testes baseada em similaridade visa encontrar uma ordem para um conjunto de testes que maximize a diferença entre eles ao longo da sua execução. Dessa forma, espera-se que diferentes partes do sistema sejam testadas e, consequentemente, diferentes defeitos sejam detectados. Neste artigo foi apresentado um algoritmo paralelo para priorização usando funções de similaridade. O algoritmo foi implementado usando OpenMPI onde um mestre e $np - 1$ escravos atuam distribuindo testes e calculando graus de similaridade em paralelo. O cálculo dos graus de similaridade é dividido entre $np - 1$ processos escravos, no algoritmo PGSM, que auxiliam posteriormente na busca paralela dos pares mais distintos, no algoritmo PLMDP. Ainda que os resultados obtidos mostrem que o algoritmo apresentado gerou uma queda no tempo médio de execução para NP maiores, o gerenciamento de recursos não se mostrou eficiente, ao analisar sua eficiência.

Referências

Cartaxo, E. G., Machado, P. D. L., and Neto, F. G. O. (2011). On the use of a similarity function for test case selection in the context of model-based testing. *Software Testing, Verif. and Reliab.*

Gîză-Belciug, F. and Pentiuc, S.-G. (2015). Parallelization of similarity matrix calculus in ontology mapping systems. In *2015 14th RoEduNet NER*, pages 50–55. IEEE.

Henard, C., Papadakis, M., Perrouin, G., Klein, J., Heymans, P., and Traon, Y. L. (2014). Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines. *IEEE Transactions on Software Engineering*, 40(7):650–670.

Open MPI (2016). Open MPI v1.8.8 documentation. `https://www.open-mpi.org/doc/v1.8/`. [Online; accessed 3-Dec-2016].

Pacheco, P. (2011). *An Introduction to Parallel Programming*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition.

Rawald, T., Sips, M., Marwan, N., and Leser, U. (2015). Massively parallel analysis of similarity matrices on heterogeneous hardware. In *Proceedings of the Workshops of the EDBT/ICDT 2015 Joint Conference (EDBT/ICDT), Brussels, Belgium, March 27th, 2015.*, pages 56–62.

Yoo, S. and Harman, M. (2012). Regression testing minimization, selection and prioritization: A survey. *Softw. Test. Verif. Reliab.*, 22(2):67–120.

Zhang, Y.-F., Tian, Y.-C., Kelly, W., and Fidge, C. (2017). Scalable and efficient data distribution for distributed computing of all-to-all comparison problems. *Future Generation Computer Systems*, 67:152 – 162.