Análise de relações entre desempenho e contenção de memória em sistemas *ccNUMA* com *kernel* Linux

Erik de Godoy Perillo[1], **Edson Borin**[1]

[1]Instituto de Computação – Universidade Estadual de Campinas (Unicamp)
Campinas/SP, Brasil.

erik.perillo@gmail.com, edson@ic.unicamp.br

Abstract. *In this work, we perform an analysis on a variety of High Performance Computing applications under different memory management policies for ccNUMA systems. We collected various metrics for considered applications via profiling. We then explored a variety of predictive models built aiming to assess the performance of new applications under different memory allocation policies from profiling data, thus identifying the best policy to be used.*

Resumo. *Neste trabalho, fazemos uma análise de diversas aplicações de Computação de Alto Desempenho sob variadas políticas de gerenciamento de memória em sistemas ccNUMA. Coletamos uma variedade de métricas pelo perfilamento das aplicações. Exploramos então diversos modelos preditivos com o fim de inferir o desempenho de novas aplicações sob as diferentes políticas de alocação de memória a partir dos dados de perfilamento das mesmas, identificando desta forma a melhor política a ser usada.*

1. Introdução

Em arquiteturas com Acesso Não-Uniforme à Memória com Coerência de *Cache*, ou *ccNUMA*[1], a latência no acesso pode variar de acordo com a posição física do banco de memória e do núcleo computacional que está acessando o dado [2], fazendo com que o posicionamento dos dados e tarefas afetem o desempenho do sistema. O sistema operacional *Linux* disponibiliza políticas de alocação de memória que podem ser usadas para melhorar o posicionamento de tarefas (processos ou *threads*) e ou dados na memória de sistemas *ccNUMA*. A política *Node Local* aloca páginas de memória no banco de memória mais próximo ao núcleo que a requisitou e a política *Interleave* aloca as páginas de forma circular entre os diversos bancos de memória do sistema para obter uma distribuição uniforme dos dados. Há também um mecanismo dedicado à migração automática de dados e tarefas, chamado de *automatic NUMA balancing* (ou *autonuma*). Este mecanismo atua durante a execução dos processos, monitorando o padrão de acessos à memória e migrando *threads* e páginas de memória para melhorar as localizações de tarefas com relação aos dados acessados por elas [4].

Uma variedade de métodos de gerenciamento de memória a serem escolhidos exige testar cada aplicação sob cada uma das políticas para identificar a que provê melhor desempenho (menor tempo de execução). Surge, então, o interesse por um método de determinar com mais eficiência qual a melhor política a ser utilizada. Métricas obtidas por perfilamento das aplicações podem prover informações sobre as características de

[1]do inglês: *cache coherent Non-Uniform Memory Access*

uso de memória das mesmas. Neste trabalho, temos dois objetivos: a) Definir um conjunto de métricas que possam fornecer informações relevantes sobre o perfil de acesso à memória de diversas aplicações de alto desempenho; b) Desenvolver um modelo preditivo que possibilite inferir o desempenho de aplicações a serem usadas no futuro com base na extração das métricas definidas em a). Considera-se tais contribuições importantes pois isso possibilitaria determinar a melhor política de memória a ser usada para uma certa aplicação com apenas uma execução da mesma, independentemente do número configurações de políticas possíveis, possibilitando então uma análise de tempo constante. Também permitiria-se ao desenvolvedor identificar quando políticas de alocação de memória podem ou não afetar significativamente o desempenho do programa em questão.

2. Materiais e métodos

Os experimentos são executados em um computador com arquitetura *ccNUMA* de 64 núcleos compostos por quatro processadores *AMD Opteron 6282* de 16 núcleos cada. Há um total de 8 nós NUMA, cada um conectado a um banco de memória de 16GB [5]. O sistema Operacional é o *Ubuntu 12.04.01 LTS* com *kernel Linux 3.19*.

Foram utilizadas três suítes de benchmarks paralelos: ***Nas Parallel Benchmarks (NPB)*** [7]: utilizada em sua versão `2.3`, compilada usando-se a suíte *gcc* na versão `4.6.3` com *OpenMP*. ***Parsec*** (subconjunto *splash2x*) [6]: as aplicações foram acionadas por meio do comando `parsecmgmt -a run -n 64 -i native -p splash2x.<APP>`, com `<APP>` dentre as aplicações da suíte. ***easy benchmarks*** [8]: as aplicações foram executadas por meio do comando `<APP> -nt 64 -nm 256 -msz 1024`, com `<APP>` dentre `add, prod, lu, ch`. Para acionar a política *Node-Local*, não há a necessidade de comandos adicionais. A política *interleave* é acionada usando-se o comando `numactl --interleave=all -- <comando>`. *Autonuma* é ativada por meio do comando `echo 1 > /proc/sys/kernel/numa_balancing`. Não usou-se *Autonuma* + *Interleave* pois notou-se em trabalhos anteriores que a combinação não traz melhoras com relação às outras configurações [11].

Para perfilar as aplicações utilizou-se a ferramenta `perf`, em sua versão `3.19.8`. Cada medição foi realizada ao menos 10 vezes. Quando o intervalo de confiança (de 95%) para cada dado excedeu 15%, novas medições foram feitas. Explicamos a seguir detalhes das métricas e os métodos de aquisição das mesmas. **Ganho de desempenho**, ou *speedup* de uma certa aplicação, é definido como quão rápido uma certa aplicação em uma certa política de alocação de memória foi com relação à mesma aplicação sob a política padrão (*Node Local*): $speedup_{pol} = \frac{time_{def}}{time_{pol}}$. O tempo considerado é o tempo real e é medido pela ferramenta *GNU time* em sua versão `1.7`. Busca-se o maior ganho de desempenho possível. **Quantidade de instruções** executadas: obtida pelo comando `perf stat -e instructions -a -c`. **Quantidade de ciclos** executados: obtida pelo comando `perf stat -e cycles -a -c`. **Ciclos em *stall* no *backend***: Quantidade de ciclos em que o *backend* do processador ficou no modo *stalled*. Geralmente, o *backend* estar parado indica que um acesso à memória está sendo realizado [10]. Métrica adquirida pelo comando `perf stat -e stalled-cycles-backend -a -c`. ***Load/Store Queue Full***: Quantidade de ciclos em que houve falta de recursos na *Load Queue*/*Store Queue* do processador. Obtidas através do comando `perf stat -e r510123 r510223`, onde `r510123` refere-se ao evento de `load` e o outro `store`. **Eventos de acesso à memória**

entre nós NUMA: Eventos de acesso à memória do nó NUMA i ao nó NUMA j (onde $0 \leq i \leq 7$ e $0 \leq j \leq 7$) de uma certa aplicação. Obtidos através do comando `perf stat -e amd_nb/event=0x1e0,umask=<m>`, onde `<m>` está em $\{0x1, 0x2, 0x4, 0x8, 0x10, 0x20, 0x40, 0x80\}$ [1], estes respectivos aos nós $0, 1, ..., 7$. **Desvio padrão de eventos de acesso à memória**: Derivada da métrica anterior. **Total de eventos de acesso à memória**: Obtido por: $mem\_acc\_sum = \sum_{i=0}^{7}\sum_{j=0}^{7} mem\_acc_{i,j}$. **Total de eventos de acesso à memória para um certo nó NUMA**: Descreve o número de acessos à memória que um certo nó NUMA sofreu. É obtida por: $mem\_acc\_to_n = \sum_{i=0}^{7} mem\_acc_{i,n}$ Onde n é o número do n-ésimo nó. **Desvio padrão de eventos de acesso à memória para um certo nó NUMA**: Derivada da métrica anterior. **Raiz quadrada da soma dos quadrados de somas de eventos de acesso à memória para um certo nó NUMA**: É obtida por: $mem\_acc\_to\_sqrtsumsq = \sqrt{\sum_{i=0}^{7} mem\_acc\_to_i{}^2}$. Essa métrica tem um valor alto quando há altas concentrações de acesso em poucos nós.

3. Análise de Dados

As primeiras tentativas na elaboração de um modelo que fosse capaz de explicar os dados e realizar predições foram feitas com base em modelos estatísticos de regressão polinomial, mas sem muito sucesso. A busca por modelos de complexidade maior, capazes de estabelecer relações não-lineares entre os dados induz ao uso de técnicas de Aprendizado de Máquina. Foram estabelecidos problemas para modelagem de regressão e classificação. O problema da regressão tem como meta prever o ganho de desempenho (de uma certa política com relação à padrão) com base nas métricas. O problema da classificação tem como meta apenas dizer qual a melhor política (a que trará maior ganho de desempenho) com base nos dados de perfilamento (*node-local*, *interleave* ou *autonuma*). Foi usada modelagem por Florestas Aleatórias [3], uma técnica que é capaz de estabelecer relações complexas entre os dados e com uma quantidade relativamente pequena de amostras, como é o caso neste trabalho, onde temos dados de 25 aplicações. O método pode ser usado em aplicações tanto de regressão quanto de classificação. O trabalho foi feito em *Python* com uso da biblioteca `sklearn 0.18.1` [9] e das classes `RandomForestRegressor` e `RandomForestClassifier`.

Todos os treinamentos de modelos foram feitos usando-se validação cruzada para a busca de melhores parâmetros. Para cada combinação de parâmetros considerada, o conjunto de aplicações foi dividido em três partes. São então treinados três preditores, usando para cada um uma das três divisões como teste para avaliar o desempenho. A combinação de parâmetros que der a melhor medida média de desempenho (R^2 no caso da regressão e acurácia no caso da classificação) entre os três conjuntos de validação cruzada é escolhida e, então o modelo final é treinado em todos os dados com esses parâmetros.

Para a regressão, o melhor estimador de ganho de desempenho com *interleave* resultou em um R^2 médio de 0.847 entre os três conjuntos de teste, com um R^2 do modelo final de 0.987. O melhor estimador de ganho de desempenho para *autonuma* resultou em um R^2 médio de 0.764 entre os conjuntos de teste, com R^2 de 0.943 para o modelo final. Para a classificação, obteve-se uma acurácia média entre os conjuntos de teste de 0.68, com acurácia do modelo final de 0.96.

4. Conclusões

O modelo preditivo baseado em Florestas Aleatórias mostrou-se capaz em prever tanto o ganho de desempenho com relação ao alívio de contenção quanto a melhor política de alocação de memória a ser usada para cada aplicação. Os resultados também permitem concluir que há relação entre o padrão de contenção no acesso à memória de uma certa aplicação com seu potencial alívio na contenção a ser provocado pelas políticas de alocação de memória alternativas, assumindo-se que as métricas colhidas refletem de alguma forma os perfis de acesso à memória das aplicações.

Um possível ponto negativo dos modelos obtidos é que estes são complexos. Perdeu-se a capacidade de inferir de forma intuitiva sobre a influência de cada dado coletado no resultado final, mas talvez não haja mesmo uma relação simples entre dados os e o desempenho. O fato da tentativa falha de análise prévia com modelos mais simples corrobora essa hipótese. Outro problema é que a quantidade de dados é relativamente pequena. Isso impede uma forma mais robusta de treinamento, o que pode acarretar em um certo vício dos preditores aqui treinados com os dados usados.

O trabalho aqui apresentado pode ser de utilidade em ambientes com arquitetura NUMA semelhantes. Com um perfilamento único de aplicações a serem executadas nesses ambientes, pode-se usar o modelo de modo a indicar a melhor política de alocação de memória, eliminando assim a necessidade de executar cada aplicação sob cada política para a determinação da melhor. Comparando-se com o método exaustivo (e as três políticas aqui discutidas), nosso método necessita de duas vezes menos execuções. Possíveis trabalhos futuros envolvem uma análise *online* de diversas etapas das aplicações e a implementação de uma ferramenta que extrai as métricas das aplicações e usa o modelo para inferir a melhor política.

Referências

[1] *BIOS and Kernel Developer's Guide (BKDG) for AMD Family 15h Models 00h-0Fh Processors*. Seção 2.4, Advanced Micro Devices, 2013.

[2] Nakul Manchanda e Karan Anand. *Non-Uniform Memory Access (NUMA)*. New York University, 4 de Maio de 2010.

[3] Stanford Statistics, *Random Forests*. url: `https://goo.gl/7XUydd`

[4] Rik van Riel e Vinod Chegu. *Automatic NUMA balancing*. Red Hat Summit, 2014.

[5] Edson Borin, Philippe Devloo, Gilvan Vieira e Nathan Shauer. *Accelerating Engineering Software on Modern Multi-Core Processors*. Unicamp, 2015.

[6] PARSEC GROUP. *Exploration of SPLASH-2X Input Sets*. Princeton University, 2011.

[7] *NAS Parallel Benchmarks*. url: `https://goo.gl/Nekzsv`

[8] *easy benchmarks*. url: `https://goo.gl/usSFMB`

[9] Sklearn Documentation. url: `https://goo.gl/xd5dmH`

[10] David Levinthal. *Performance Analysis Guide*. 2008.

[11] Erik Perillo, Gilvan Vieira, Philippe Devloo e Edson Borin. *Uma Análise de Desempenho do Mecanismo Automático de Balanceamento NUMA do Linux*. url: `https://goo.gl/g5N5aq`. 2015.