Uma Análise de Desempenho do Mecanismo Automático de Balanceamento NUMA do Linux

Erik G. Perillo[1], Gilvan S. Vieira[1], Philippe R. B. Devloo[2], Edson Borin[1]

[1]Instituto de Computação – UNICAMP

[2]Faculdade de Engenharia Civil, Arquitetura e Urbanismo – UNICAMP

erik.perillo@gmail.com, gilvandsv@gmail.com

phil@fec.unicamp.br, edson@ic.unicamp.br

Resumo. *O mecanismo automático de balanceamento NUMA do Linux, ou* automatic NUMA Balancing*, realiza a migração de dados e* threads *para reduzir a latência dos acessos à memória. Neste trabalho, nós avaliamos o impacto deste mecanismo e da política de alocação de memória* Interleave *na execução do* benchmark *paralelo NAS em uma máquina NUMA com 64 núcleos computacionais. Nossos resultados indicam que os desempenhos de aplicações paralelas podem sensíveis à localidade dos dados e tarefas e que tanto o mecanismo automático de balanceamento NUMA quanto a política de alocação* Interleave *podem ser usados para melhorar a distribuição dos dados e, consequentemente, o desempenho do sistema.*

1. Introdução

As primeiras arquiteturas *multicore* possuíam dois ou mais núcleos conectados à memória principal. Apesar dos núcleos computacionais serem duplicados, estes núcleos liam e escreviam dados através de um único barramento ou controlador de memória. Em função disto, quando vários núcleos faziam acesso à memória ao mesmo tempo, havia contenção no acesso o que poderia tornar este mecanismo um gargalo [Borin et al. 2014].

Para permitir que o desempenho escale junto com o número de núcleos, arquiteturas de Multiprocessamento Simétrico, ou SMP[1], estão sendo substituídas por arquiteturas com Acesso Não-Uniforme à Memória com coerência de *cache*, ou ccNUMA[2], onde a memória é distribuída fisicamente em múltiplos bancos e controladores de memória.

Da mesma forma que nas arquiteturas SMP, as arquiteturas ccNUMA permitem que código executando em qualquer núcleo acesse qualquer palavra de memória. No entanto, os bancos de memória estão distribuídos e são controlados por múltiplos controladores de memória, possibilitando que núcleos acessem de forma concorrente os diferentes bancos de memória. Como resultado, a latência no acesso pode variar de acordo a posição do banco de memória e do núcleo computacional no sistema, fazendo com que o posicionamento dos dados e tarefas afetem o desempenho do sistema.

Diversas abordagens foram propostas para melhorar o posicionamento de tarefas (processos ou *threads*) e ou dados na memória de sistemas ccNUMA, reduzindo assim a latência no acesso e a concorrência por bancos de memória [Piccoli et al. 2014,

[1]do inglês: *Symmetric Multi Processing*

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

Borin et al. 2014]. O sistema operacional Linux, por exemplo, disponibiliza políticas de alocação de memória que podem ser usadas para melhorar a localidade ou a distribuição de dados em sistemas ccNUMA. Dentre elas, há a *Node Local*, que aloca páginas de memória no banco de memória mais próximo ao núcleo computacional que a requisitou e a política *Interleave*, que aloca as páginas de forma circular entre os diversos bancos de memória do sistema para obter uma distribuição de uniforme dos dados.

Além destas políticas de alocação de memória, um mecanismo dedicado à migração automática de dados e tarefas em sistemas ccNUMA foi introduzido recentemente no núcleo do sistema operacional Linux. Este mecanismo, chamado de *automatic NUMA balancing*, já está habilitado por padrão nos sistemas mais recentes e 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.

Neste trabalho, utilizamos a suíte de aplicações paralelas NAS *Parallel Benchmarks* para avaliar o impacto das políticas de alocação de memória e do mecanismo automático de balanceamento NUMA do Linux no desempenho de aplicações paralelas executadas em um sistema ccNUMA. Além disso, discutimos os parâmetros configuráveis deste mecanismo e mostramos como os mesmos podem ser ajustados para se obter melhores resultados de desempenho. As contribuições deste trabalho são:

- Nós explicamos o funcionamento do mecanismo automático de balanceamento NUMA do Linux e apresentamos uma avaliação do seu desempenho em um sistema ccNUMA.
- Nós mostramos que, em diversas situações, a política de alocação de memória *Interleave*, apesar de ser mais simples, ainda provê resultados de desempenho melhores do que o mecanismo automático de balanceamento NUMA, introduzido recentemente no núcleo do sistema operacional Linux.
- Nós mostramos que, em alguns casos, o desempenho do mecanismo automático de balanceamento NUMA do Linux pode ser melhorado através do ajuste de seus parâmetros de configuração.
- Nós apresentamos resultados que indicam uma forte correlação entre o potencial de ganho de desempenho e a taxa de acessos à memória das aplicações.

O texto está organizado da seguinte forma: na Seção 2 apresentamos uma visão geral do processo de alocação de memória do Linux em sistemas ccNUMA, na Seção 3 discutimos o funcionamento do mecanismo automático de balanceamento NUMA do Linux, na Seção 4 apresentamos as condições de teste e a metodologia utilizada. Seção 5 apresentamos os resultados experimentais, na Seção 6 apresentamos os trabalhos relacionados e, por fim, na Seção 7 apresentamos as conclusões de nossa pesquisa.

2. Alocação de Memória em Sistemas ccNUMA

Nos sistemas ccNUMA, os bancos de memória são controlados por múltiplos controladores de memória, permitindo assim um maior grau de paralelismo no acesso aos dados da memória. Cada controlador de memória, também conhecido como NorthBridge em arquiteturas AMD, está associado a um subconjunto dos núcleos computacionais do sistema, formando um nó NUMA. Acessos à memória realizados por controladores de memória que estão dentro do mesmo nó NUMA são considerados acessos locais enquanto que

acessos realizados por controladores em outros nós NUMA são considerados acessos remotos.

Os acessos remotos são realizados através de circuitos que conectam os nós NUMA, o que ocasiona uma maior latência no acesso. Apesar da latência variar de acordo com a posição do núcleo computacional e do controlador de memória que está atendendo à requisição, o acesso é transparente para o *software*, já que o *hardware* encaminha a requisição de acesso para o controlador de memória correto com base no endereço físico da requisição. Além disso, a memória *cache* de todos os núcleos computacionais no sistema mantêm-se sempre coerente por meio de controladores especiais em cada nó.

Apesar de permitir um maior grau de paralelismo no acesso aos bancos de memória, os acessos remotos podem ser significativamente mais lentos do que os acessos locais e uma má distribuição dos dados nos bancos de memória pode causar uma sobrecarga de desempenho no sistema. Além disso, a concentração dos dados em bancos de memória associados a um único controlador de memória pode causar os mesmos problemas de congestionamento de dados encontrados em sistemas SMP. Dessa forma, o posicionamento dos dados e das tarefas é um fator importante para o desempenho de sistemas ccNUMA.

No sistema operacional Linux as páginas de memória são alocadas fisicamente sob demanda. Nesta abordagem, chamada *first-touch*, as páginas do processo ainda não tocadas (lidas ou escritas) não possuem memória física reservada. Assim, uma página somente será alocada fisicamente quando o processo a tocar pela primeira vez. Neste momento, a unidade de gerenciamento de memória do processador produzirá uma interrupção indicando a falta de página e o sistema operacional será responsável por alocar a memória física para a página. Em sistemas ccNUMA o sistema operacional também deve selecionar o banco de memória que irá fornecer a memória física. Esta escolha é determinada pela política de memória do sistema, processo ou região da memória. As duas principais políticas de alocação de memória no Linux são:

- *Node Local*: a memória física é alocada em bancos de memória próximos fisicamente ao núcleo computacional que causou a falta de página, ou seja, dentro do mesmo nó NUMA. Esta política visa reduzir o número de acessos remotos posicionando os dados no banco de memória local ao núcleo computacional que está executando a tarefa naquele momento.
- *Interleave*: as páginas de memória são alocadas de forma circular entre os bancos de memória de diferentes nós. Por exemplo, numa máquina com dois nós, a primeira página será alocada nos bancos de memória do nó 0, a segunda nos bancos do nó 1, a terceira novamente nos bancos do nó 0 e assim por diante. Nesta política, o objetivo é distribuir dos dados de forma uniforme entre os diferentes bancos de memória do sistema, reduzindo a chance de o mesmo concentrar todos os acessos num único controlador de memória.

A política de alocação *Node Local* pode gerar bons resultados se as tarefas que inicializam os dados são as mesmas que irão realizar o processamento. Por outro lado, caso a tarefa que tocou a página pela primeira vez não seja aquela que irá usar os dados da mesma, há um risco maior dos acessos serem remotos. Esta situação ocorre frequentemente em aplicações paralelas, pois programadores tendem a criar uma fase de

inicialização dos dados enquanto as tarefas que realizarão a computação ainda não foram criadas [Broquedis et al. 2010].

3. O Mecanismo Automático de Balanceamento NUMA do Linux

Na versão 3.8 do *kernel* do Linux foi adicionada a base para implementação do mecanismo automático de balanceamento NUMA, ou *automatic NUMA balancing*. Este mecanismo detecta acessos à memória através de faltas de página induzidas pelo sistema operacional [Gorman 2012]. A partir destas informações, o sistema migra páginas e/ou tarefas (*thread* ou processo) entre nós NUMA para reduzir a latência no acesso à memória e balancear a carga de trabalho entre os nós.

Para se alcançar essas metas, duas estratégias são utilizadas. Na primeira, CPU *follows memory*, busca-se alocar a tarefa no mesmo nó onde ocorre a maior parte de seus acessos à memória. Na segunda, chamada *memory follows* CPU, quando a tarefa se encontra estável em um nó, busca-se mover o restante das páginas de memória por ela acessada remotamente para o banco de memória local.

As informações requeridas para o funcionamento do mecanismo são coletadas pelo monitoramento de tarefas, feito por meio de faltas de páginas induzidas pelo sistema operacional, chamadas de NUMA *hinting page faults*. Elas são obtidas marcando-se as entradas na tabela de mapeamento de páginas desejadas contra escrita e leitura. Assim, pode-se detectar quando uma tarefa requisita aquela página pela falta de página que será gerada na tentativa de acesso à mesma.

Com informações adquiridas pelas *hinting page faults*, elaboram-se estatísticas sobre cada tarefa e seus acessos a cada nó do sistema. Com esses dados, é possível determinar se a localização atual da tarefa pode ser otimizada, ou seja, se há outro nó no sistema para o qual a tarefa possa ser movida de forma que se maximize o número de acessos locais à memória. É esperado que existam múltiplas tarefas sendo executadas ao mesmo tempo no sistema, então sempre é feita uma comparação de modo a se determinar se a mudança de uma tarefa ou até mesmo a troca de nós entre duas tarefas proverá um ganho real de desempenho ao sistema.

São também formados grupos NUMA que representam tarefas relacionadas ou por estarem em uma mesma CPU ou por serem de um mesmo processo. As estatísticas e critérios de migração para grupos NUMA são análogos aos aplicados a tarefas separadas.

Quanto à política *memory follows* CPU, há alguns critérios para que páginas sejam migradas com eficiência. Usa-se um filtro quadrático, ou seja, a migração será considerada quando houver duas faltas de página seguidas referenciando uma mesma área de memória. O tipo de falta de página também é determinante. Se for um *private page fault*, ou seja, a mesma tarefa acessou a página duas vezes seguidas, a página é migrada sem as considerações adicionais que uma *shared page fault* geraria, ou seja, avaliar se a migração para o nó pretendido não afetaria negativamente o desempenho do sistema por conta de outras tarefas que acessam aquela página.

O mecanismo automático de balanceamento NUMA funciona com uma certa frequência, sendo esta adaptada de acordo com o estado atual do sistema. Se as tarefas estiverem acessando dados em páginas dos bancos de memória locais, de forma geral, a frequência de monitoramento dos acessos é reduzida. Por outro lado, se os acessos

aos nós remotos são frequentes, então o sistema aumenta a frequência de monitoramento dos acessos, produzindo assim mais informações para realizar a migração de tarefas ou páginas. A frequência mínima e máxima de monitoramento dos acessos pode ser ajustada, assim como outros parâmetros listados a seguir:

- `numa_balancing_scan_delay_ms`: Determina o tempo inicial de pausas entre o monitoramento dos acessos. Como discutido anteriormente, o intervalo entre o monitoramento é ajustado dinamicamente em função do número de acessos locais e remotos. Apesar de ser automaticamente ajustável, seus valores mínimos e máximos podem ser especificados pelo usuário.
- `numa_balancing_scan_period_min_ms`: Determina o intervalo mínimo entre o monitoramento de acessos à memória.
- `numa_balancing_scan_period_max_ms`: Determina o intervalo máximo entre o monitoramento de acessos à memória.
- `numa_balancing_scan_size_mb`: Determina a quantidade de páginas (em MB) que serão analisadas durante cada monitoramento.
- `numa_balancing_settle_count`: Determina por quantos períodos o mecanismo vai manter uma tarefa em um nó que foi escolhido até que ela possa ser migrada novamente, se necessário.
- `numa_balancing_migrate_deferred`: Determina quantas vezes a migração de páginas compartilhadas deverá ser postergada após uma migração de página compartilhada for ignorada por causa de um *shared page fault*.

4. Metodologia

Nesta seção apresentamos o ambiente e modo com o qual os experimentos foram realizados.

Utilizamos em nossos experimentos um computador de 64 núcleos computacionais com arquitetura ccNUMA composto por quatro processadores AMD Opteron 6282 de 16 núcleos cada. Cada processador é composto por dois dispositivos de circuitos integrados (nós) contendo oito núcleos AMD64 superescalares. Além de uma North-Bridge, responsável por encaminhar dados entre as interfaces de comunicação. Utilizamos também o sistema operacional Linux Ubuntu 12.04.01 LTS com kernel 3.13.

Para avaliar as diferentes técnicas utilizamos a suíte de *benchmarks* NPB (*NAS Parallel Benchmarks*) na versão OpenMP. Os *benchmarks* NAS foram desenvolvidos com o propósito de analisar o desempenho de grandes computadores com processamento paralelo [Bailey et al. 1994]. São sete aplicações:

- EP - *Embarrassingly Parallel*
- CG - *Conjugate Gradient, irregular memory access and communication*
- MG - *Multi-Grid on a sequence of meshes, long- and short-distance communication, memory intensive*
- FT - *discrete 3D fast Fourier Transform, all-to-all communication*
- BT - *Block Tri-diagonal solver*
- SP - *Scalar Penta-diagonal solver*
- LU - *Lower-Upper Gauss-Seidel solver*

Os *benchmarks* foram compilados utilizando-se a suíte gcc na versão 4.6.3 com *flags* de otimização `-mcmodel=large -shared -fopenmp -O3`.

Também utilizamos a ferramenta `numactl` para fixar as *threads* nos nós NUMA e para mudar a política de alocação de memória. Já para acessar os contadores de *hardware* utilizamos as ferramentas do pacote `likwid` na versão 3.0.

O ganho de desempenho, métrica utilizada para as comparações dos resultados, diz respeito a quantas vezes menor foi o tempo real de execução de uma aplicação em certa configuração em relação ao tempo da mesma aplicação com as configurações padrões do sistema operacional.

Os resultados foram obtidos a partir da execução das aplicações do *benchmark* com as seguintes configurações de sistema:

- *Interleave*: o modo de alocação de memória *Interleave* foi utilizado. Lembrando que o modo padrão é o *Node Local*.
- *Balanceamento NUMA*: o mecanismo automático de balanceamento NUMA do sistema está habilitado.

  As duas configurações nunca foram ativadas simultaneamente.

Além dos resultados obtidos com a configuração padrão dos parâmetros do mecanismo automático de balanceamento NUMA, investigamos o impacto dos parâmetros do mecanismo no desempenho de cada aplicação. Em um primeiro momento, foi feita uma análise da influência de cada um deles. Os que se mostraram relevantes para as aplicações foram aqueles que regulam a frequência com que os acessos à memória são monitorados no processo de balanceamento NUMA. Tal frequência está associada ao *scan rate*, que é a razão entre o tamanho da memória a ser monitorada e o intervalo de tempo entre os monitoramentos. Como o que dita o intervalo é o `scan_delay` e este, por sua vez, é adaptativo, o que se obtém é uma faixa de *scan rate*, *scan rate min* e *scan rate max*, determinados pelos parâmetros `numa_balancing_scan_period_min_ms` e `numa_balancing_scan_period_max_ms`.

Para se analisar os impactos das variações de *scan rate*, as aplicações foram testadas com os parâmetros `numa_balancing_scan_period_min_ms` e `numa_balancing_scan_period_max_ms` nos seguintes valores, respectivamente:

- 0 e 50 ms;
- 100 e 500 ms;
- 500 e 1000 ms;
- 1000 e 2000 ms;
- 4000 e 6000 ms;
- 6000 e 10000 ms;

com o parâmetro `numa_balancing_scan_delay_ms` sempre como o valor médio entre os extremos.

Além disso, para uma faixa de período de tempo entre monitoramento de acessos curta (entre 0 e 100ms), variou-se o parâmetro `numa_balancing_scan_size_mb` para os seguintes valores:

- 32 MB;
- 64 MB;
- 128 MB;

Avaliamos o desempenho de todas as aplicações para cada uma destas configurações e selecionamos a melhor configuração para cada aplicação.

5. Resultados Experimentais

Nesta seção apresentamos os resultados obtidos em nossos experimentos e uma análise dos mesmos.

5.1. Configuração Padrão

A Figura 1 mostra os ganhos desempenho das configurações em relação à política de alocação padrão (*Node Local*) do Linux. Nela podemos notar que, na maior parte das aplicações, a política de alocação *Interleave* apresenta maiores ganhos de desempenho do que o mecanismo automático de balanceamento NUMA, com resultado diferente apenas para a aplicação `ft.C`.

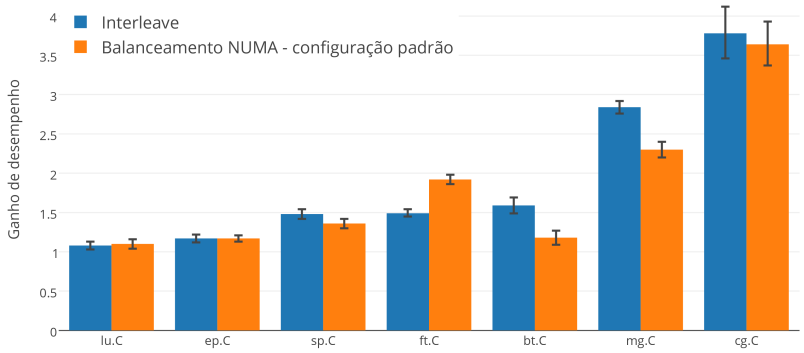

Figura 1. Ganhos de desempenho com a política *Interleave* e como o mecanismo automático de balanceamento NUMA.

Os ganhos de desempenho da política *Interleave* podem ser explicados pelo fato desta política distribuir as páginas do programa entre os bancos de memória, evitando assim o acúmulo dos dados em apenas um dos bancos de memória, o que reduziria o paralelismo no acesso durante a execução das múltiplas *threads*. Note, no entanto, que a heurística não visa melhorar a localidade no acesso aos dados e seu uso em aplicações não paralelas pode reduzir o desempenho do sistema.

O mecanismo automático de balanceamento NUMA migra páginas e tarefas com o intuito de aumentar o paralelismo no acesso aos dados e reduzir a latência no acesso através da co-alocação de tarefas e dados frequentemente acessados dentro do mesmo nó. Dessa forma, é esperado que o desempenho deste mecanismo seja melhor do que o da política *Interleave*, já que o mesmo também melhora a localidade no acesso aos dados. Entretanto, o processo de monitoramento dos acessos pode adicionar uma sobrecarga significativa no tempo de execução, caso a frequência de monitoramento seja alta. Por outro lado, uma baixa frequência de monitoramento pode fazer com que o sistema demore para movimentar os dados e as tarefas, permitindo que o sistema execute com baixo desempenho devido a uma localização ruim dos dados e das tarefas. Na próxima seção, discutiremos como podemos ajustar os parâmetros do mecanismo automático de balanceamento NUMA do Linux para minimizar estes problemas.

5.2. Busca pela melhor configuração

A Figura 2 mostra os resultados de ganho de desempenho com algumas das configurações em relação à política de alocação padrão. São exibidos os resultados com a política *Interleave*, com o balanceamento automático NUMA em suas configurações: padrão, com um *scan rate* consideravelmente maior que o padrão e com a configuração particular em que houve melhor resultado para cada aplicação.

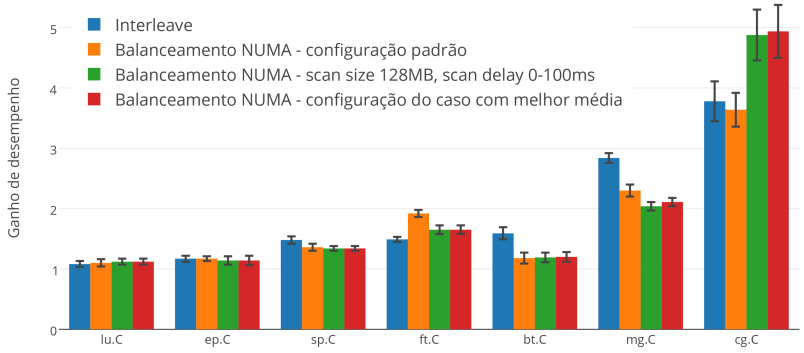

Figura 2. Ganho de desempenho com ajustes dos parâmetros do mecanismo automático de balanceamento NUMA.

Nota-se que, em geral, a variação do *scan rate* não gerou resultados significativamente melhores com relação à configuração padrão do mecanismo automático de balanceamento NUMA. Para a aplicação `cg.C`, entretanto, houve um grande ganho de desempenho com o aumento do *scan rate*, chegando a 4,94x com relação à política de alocação de memória padrão.

Podemos observar também que uma configuração que é boa para uma determinada aplicação nem sempre é boa para outra aplicação. Este é o caso da configuração com o *scan size* igual a 128 MB e *scan delay* entre 0 e 100 ms, que obteve resultados melhores do que a configuração padrão para a aplicação `cg.C` e resultados piores do que a configuração padrão para as aplicações `ft.C` e `mg.C`.

5.3. Relação entre ganhos de desempenho e acessos à memória

Os resultados das Figuras 1 e 2 indicam que, apesar do alto grau de paralelismo[3], o desempenho de algumas aplicações não é afetado significativamente pelo reposicionamento dos dados e das tarefas. Para explicar esta diferença nós analisamos a quantidade de acessos à memória que cada aplicação realiza e discutimos o seu impacto no desempenho da aplicação.

Processadores modernos possuem *caches* de dados em *hardware* para reduzir a latência e aumentar o paralelismo no acesso aos dados. Quando os dados se encontram na *cache* do processador os núcleos computacionais não precisam acessar a memória

[3]Todas as aplicações utilizaram 64 *threads* durante a execução.

e, consequentemente, não precisam competir pelos controladores de memória. Nestas situações, o reposicionamento das páginas e/ou tarefas do sistema afetam muito pouco o desempenho pois a quantidade de tempo adicionada à execução do programa em função dos poucos acessos remotos ou contenção causada por estes acessos é pequena e qualquer melhora nesta métrica não afeta significativamente o tempo total de execução do programa.

Fazendo uso da unidade de monitoramento de desempenho das CPUs e dos controladores de memória do sistema, nós medimos a quantidade de acessos à memória e o número de instruções executadas por cada uma das aplicações e, com isto, calculamos a taxa de acessos à memória por mil instruções. Esta taxa indica a frequência com que o programa acessa a memória e um baixo valor sugere que o desempenho no acesso à memória não causa muito impacto no desempenho geral do programa. De fato, como podemos ver na Figura 3, há uma relação forte entre os ganhos obtidos com as técnicas de reposicionamento das páginas e/ou tarefas e a taxa de acessos à memória por kilo instruções.

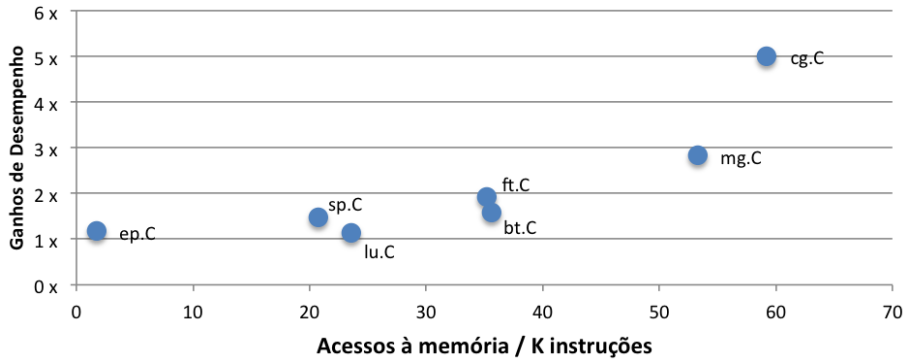

Figura 3. Ganhos de desempenho *versus* acessos à memória por kilo instrução.

Os resultados da Figura 3 sugerem ainda que há uma relação quadrática entre a taxa de acessos à memória e o potencial de ganho de desempenho com o reposicionamento de dados e tarefas neste sistema NUMA.

5.4. Melhorias na localidade dos acessos à memória

Nesta seção nós apresentamos o impacto que o mecanismo automático de balanceamento NUMA e a política de alocação *Interleave* causam na localidade de acessos à memória. Neste experimento, nós utilizamos os contadores de eventos da NorthBridge de cada nó para medir o número de acessos a bancos de memória locais e remotos durante a execução dos *benchmarks*.

lu.C O mecanismo de balanceamento NUMA do Linux melhorou o desempenho da aplicação lu.C em 10% em relação à alocação local, e a política de alocação *Interleave* melhorou em 8%. A utilização concomitante de ambos os mecanismos não apresentou ganho superior.

O mapa de acesso à memória da Figura 4 apresenta uma visão geral dos acessos à memória quando a aplicação lu.C é executada com a alocação *Node Local*, a alocação *Interleave* e com o mecanismo automático de balanceamento NUMA. O mapa é uma

matriz que indica a porcentagem dos acessos que foram realizados entre cada par de nós NUMA. Por exemplo, a segunda célula da primeira matriz (*Node Local*) indica que 9% dos acessos à memória foram realizados por núcleos computacionais do nó 0 para os bancos de memória pertencentes ao nó 1 quando a aplicação é executada com este mecanismo de alocação de memória. As células da diagonal da matriz representam os acessos locais enquanto que as restantes representam os acessos remotos. A porcentagem total de acessos locais e remotos é indicada logo abaixo da matriz. A tabela também foi colorida de forma a destacar porcentagens maiores com cores mais quentes. A última linha da tabela indicada a concentração de acessos nos bancos de memória em cada um dos nós NUMA do sistema. Por exemplo, 38% dos acessos à memória foram realizados nos bancos de memória do nó 1 quando a aplicação é executada com a política padrão (*Node Local*) de alocação de memória.

	Node Local				*Interleave*				Balanceamento NUMA			
De/Para	Nó 0	Nó 1	Nó 2	Nó 3	Nó 0	Nó 1	Nó 2	Nó 3	Nó 0	Nó 1	Nó 2	Nó 3
Núcleo no nó 0	6%	9%	6%	4%	6%	6%	6%	7%	11%	9%	3%	3%
Núcleo no nó 1	4%	10%	5%	4%	6%	6%	6%	6%	6%	13%	3%	3%
Núcleo no nó 2	5%	8%	6%	6%	6%	6%	7%	7%	5%	7%	9%	3%
Núcleo no nó 3	5%	10%	5%	5%	6%	6%	7%	7%	5%	8%	2%	10%
Todos os núcleos	20%	38%	23%	19%	24%	23%	26%	27%	28%	37%	17%	18%
Local	26%				25%				43%			
Remoto	74%				75%				57%			

Figura 4. Mapa de acessos à memória para a aplicação lu.C.

Os resultados do mapa da Figura 4 indicam que 74% dos acessos à memória são feitos a bancos de memória de outros nós (remotos) quando se utiliza a política padrão de alocação. Já com o mecanismo de balanceamento NUMA, há uma redução de acessos remotos de 74% para 57%. Contudo, ainda há uma concentração de acessos no nó 1. Este resultado sugere que a localidade de dados e tarefas poderia ser melhorada ainda mais, entretanto, não obtivemos melhora significativa com a busca por melhores parâmetros para o mecanismo automático de balanceamento NUMA para esta aplicação.

ep.C: O mapa de acessos da Figura 5 indica que há uma redução significativa no número de acessos remotos quando o mecanismo automático de balanceamento NUMA é usado na aplicação ep.C. Entretanto, apesar da média dos tempos de execução indicarem uma pequena melhora no desempenho na aplicação ep.C, os intervalos de confiança das amostras indicam que não há significância estatística nas diferenças entre as médias. Isto, pode ser explicado pela baixa (a menor) taxa de acessos à memória por instruções executadas, discutida na Seção 5.3.

	Node Local				*Interleave*				Balanceamento NUMA			
De/Para	Nó 0	Nó 1	Nó 2	Nó 3	Nó 0	Nó 1	Nó 2	Nó 3	Nó 0	Nó 1	Nó 2	Nó 3
Núcleo no nó 0	1%	18%	0%	8%	6%	5%	5%	6%	18%	3%	0%	2%
Núcleo no nó 1	0%	17%	0%	7%	9%	8%	8%	10%	1%	28%	0%	3%
Núcleo no nó 2	0%	16%	0%	7%	4%	3%	4%	8%	1%	3%	11%	6%
Núcleo no nó 3	0%	18%	0%	8%	6%	5%	5%	8%	1%	4%	0%	19%
Todos os núcleos	2%	68%	0%	30%	24%	22%	22%	32%	21%	38%	11%	30%
Local	26%				26%				77%			
Remoto	74%				74%				23%			

Figura 5. Mapa de acessos à memória para a aplicação ep.C.

sp.C: O mapa de acessos da aplicação sp.C, apresentado na Figura 6, mostra que o uso da política *Interleave* produz a mesma porcentagem de acessos locais (25%) e remotos

(75%) que o uso da política de alocação padrão, enquanto que o mecanismo automático de balanceamento NUMA melhora a porcentagem dos acessos locais (44%).

	Node Local				*Interleave*				Balanceamento NUMA			
De/Para	Nó 0	Nó 1	Nó 2	Nó 3	Nó 0	Nó 1	Nó 2	Nó 3	Nó 0	Nó 1	Nó 2	Nó 3
Núcleo no nó 0	7%	15%	3%	1%	5%	6%	5%	6%	13%	4%	2%	6%
Núcleo no nó 1	6%	13%	3%	1%	6%	7%	6%	7%	7%	10%	2%	6%
Núcleo no nó 2	5%	14%	4%	1%	6%	7%	6%	7%	6%	2%	11%	4%
Núcleo no nó 3	6%	15%	3%	2%	6%	7%	6%	7%	9%	4%	4%	10%
Todos os núcleos	23%	58%	14%	6%	24%	26%	22%	27%	35%	21%	19%	26%
Local	25%				25%				44%			
Remoto	75%				75%				56%			

Figura 6. Mapa de acessos à memória para a aplicação sp.C.

Apesar da menor porcentagem de acessos locais, a política *Interleave* provê o melhor resultado de desempenho, sendo 1,48x melhor do que a política padrão enquanto que o balanceamento NUMA é apenas 1,38x melhor. Um dos fatores que causam esta diferença é a contenção causada pela concentração de acessos a bancos de memória em um mesmo nó. Essa concentração, indicada pelo somatório das colunas da tabela é resumida na última linha da tabela. Observe que, apesar de uma menor porcentagem de acessos remotos, o mecanismo de balanceamento NUMA fez com que 35% dos acessos fossem concentrados no nó 0, enquanto que a política *Interleave* foi capaz de distribuir de forma regular os acessos aos bancos de memória de diferentes nós.

bt.C: Da mesma forma que na aplicação sp.C, a política de alocação *Interleave* proveu o melhor ganho de desempenho para a aplicação bt.C, sendo 1,59x melhor do que a política padrão enquanto que o mecanismo de balanceamento NUMA é apenas 1,18x melhor. Entretanto, como podemos ver na Figura 7, além de melhorar a porcentagem de acessos locais para a aplicação bt.C, o mecanismo de balanceamento NUMA foi capaz de melhorar a distribuição dos acessos aos diferentes bancos de memória.

	Node Local				*Interleave*				Balanceamento NUMA			
De/Para	Nó 0	Nó 1	Nó 2	Nó 3	Nó 0	Nó 1	Nó 2	Nó 3	Nó 0	Nó 1	Nó 2	Nó 3
Núcleo no nó 0	6%	7%	4%	7%	6%	6%	6%	6%	14%	3%	3%	3%
Núcleo no nó 1	7%	6%	4%	7%	6%	6%	6%	6%	3%	15%	3%	3%
Núcleo no nó 2	8%	7%	4%	7%	7%	7%	6%	6%	5%	3%	13%	4%
Núcleo no nó 3	8%	6%	4%	7%	8%	6%	6%	6%	5%	4%	3%	13%
Todos os núcleos	30%	26%	16%	28%	27%	25%	24%	24%	28%	25%	22%	24%
Local	23%				23%				56%			
Remoto	77%				77%				44%			

Figura 7. Mapa de acessos à memória para a aplicação bt.C.

ft.C: Para a aplicação ft.C, o uso do mecanismo automático de balanceamento NUMA do Linux proveu um ganho de desempenho de 1,92x, enquanto que o uso da política *Interleave* proveu um ganho de apenas 1,49x. O mapa de acessos da Figura 8 indica que, apesar da distribuição regular das páginas entre os bancos de memória, há uma concentração dos acessos à memória no nó 2 quando a política *Interleave* é usada. Por outro lado, o mecanismo de balanceamento NUMA consegue reduzir a porcentagem de acessos remotos de 66% para 44% além de balancear a carga de acessos entre os diferentes nós.

mg.C: Como podemos ver na Figura 9, além de melhorar a porcentagem de acessos locais, o mecanismo de balanceamento NUMA foi capaz de melhorar a distribuição dos

	Node Local				*Interleave*				Balanceamento NUMA			
De/Para	Nó 0	Nó 1	Nó 2	Nó 3	Nó 0	Nó 1	Nó 2	Nó 3	Nó 0	Nó 1	Nó 2	Nó 3
Núcleo no nó 0	5%	11%	4%	3%	4%	5%	9%	5%	12%	4%	5%	2%
Núcleo no nó 1	3%	16%	4%	3%	5%	4%	10%	5%	2%	16%	6%	2%
Núcleo no nó 2	3%	12%	7%	1%	6%	5%	10%	5%	3%	4%	15%	2%
Núcleo no nó 3	2%	16%	4%	7%	6%	5%	11%	4%	3%	5%	5%	13%
Todos os núcleos	12%	55%	19%	14%	21%	19%	40%	19%	20%	28%	32%	20%
Local	34%				23%				56%			
Remoto	66%				77%				44%			

Figura 8. Mapa de acessos à memória para a aplicação ft.C.

acessos aos diferentes bancos de memória. Entretanto, assim como na aplicação bt.C, o uso da política de alocação *Interleave* forneceu um ganho de 2,84x contra 2,3x do mecanismo de balanceamento NUMA. Como discutido antes, o mecanismo automático de balanceamento NUMA do Linux induz faltas de páginas para monitorar os acessos à memória. O excesso de faltas de páginas pode causar uma sobrecarga de tempo excessiva, reduzindo os benefícios obtidos com o mecanismo. De fato, uma análise com a ferramenta `perf stat` revelou que o mecanismo de balanceamento NUMA causa um grande número de faltas de páginas nesta aplicação.

	Node Local				*Interleave*				Balanceamento NUMA			
De/Para	Nó 0	Nó 1	Nó 2	Nó 3	Nó 0	Nó 1	Nó 2	Nó 3	Nó 0	Nó 1	Nó 2	Nó 3
Núcleo no nó 0	1%	21%	0%	0%	6%	6%	6%	6%	16%	4%	2%	1%
Núcleo no nó 1	0%	19%	0%	0%	6%	6%	6%	6%	3%	21%	1%	1%
Núcleo no nó 2	0%	27%	0%	0%	7%	7%	6%	6%	5%	3%	16%	1%
Núcleo no nó 3	0%	30%	0%	0%	8%	6%	6%	6%	3%	3%	1%	19%
Todos os núcleos	2%	97%	1%	1%	27%	25%	24%	24%	27%	31%	20%	23%
Local	21%				24%				71%			
Remoto	79%				76%				29%			

Figura 9. Mapa de acessos à memória para a aplicação mg.C.

cg.C: Para esta aplicação, a política de alocação *Interleave* também ofereceu um ganho de desempenho superior (3,78x) ao obtido com o mecanismo automático de balanceamento NUMA do Linux (3,64x). Entretanto, como discutido na Seção 5.2, um ajuste nos parâmetros do mecanismo de balanceamento NUMA do Linux permitiu um ganho de desempenho de 5,01x.

As aplicações cg.C e mg.C foram as que mais se beneficiaram com as técnicas de distribuição e migração de dados, obtendo ganhos de desempenho de 5,01x e 2,84x, respectivamente. Além da alta taxa de acessos à memória, como discutido na Seção 5.3, ambas possuem uma concentração muito grande (97%) de acessos aos bancos de memória de um único nó, como pode ser visto nas Figuras 9 e 10. Tal número indica que, para essas aplicações, há um alto número de acessos remotos à memória e, como estes são concentrados em um único nó, a contenção no acesso à memória torna-se prejudicial ao desempenho.

6. Trabalhos Relacionados

Vários trabalhos na literatura apresentaram soluções para o problema de localidade no acesso à memória em arquiteturas ccNUMA. Alguns destes trabalhos, focam na distribuição manual dos dados pelo programadores. Wittmann e Hager [Wittmann and Hager 2010] propõem uma camada de *software* para reduzir o número

	Node Local				*Interleave*				Balanceamento NUMA			
De/Para	Nó 0	Nó 1	Nó 2	Nó 3	Nó 0	Nó 1	Nó 2	Nó 3	Nó 0	Nó 1	Nó 2	Nó 3
Núcleo no nó 0	18%	1%	0%	0%	5%	5%	5%	7%	18%	5%	0%	1%
Núcleo no nó 1	19%	0%	0%	0%	6%	5%	6%	7%	4%	17%	2%	1%
Núcleo no nó 2	29%	0%	0%	0%	8%	5%	5%	9%	1%	6%	17%	1%
Núcleo no nó 3	31%	0%	0%	0%	8%	6%	6%	8%	1%	9%	1%	16%
Todos os núcleos	97%	1%	1%	1%	26%	21%	21%	31%	23%	37%	21%	19%
Local	19%				24%				68%			
Remoto	81%				76%				32%			

Figura 10. Mapa de acessos à memória para a aplicação cg.C.

de acessos remotos co-alocando tarefas e dados em filas de localidade para cada nó NUMA.

Yang *et al.* [Yang et al. 2009] mostram que a migração dinâmica de páginas manual em conjunto a política *Interleave* pode fornecer bons ganhos de desempenho para a aplicação química Gaussian O3. Da mesma forma, Norden *et al.* [Nordén et al. 2008] analisam e otimizam o acesso à memória num sistema ccNUMA utilizando chamadas do sistema operacional Solaris para migração de páginas.

Majo e Gross [Majo and Gross 2013] avaliam o impacto da combinação de transformações de código com a utilização da política de distribuição de memória *Interleave*. Os resultados com as aplicações `streamcluster` e `ferret` do *benchmark* PARSEC mostraram ganhos de desempenho de 21% e 35% apenas com a utilização da política *Interleave*. Estes ganhos se devem ao balanceamento de carga entre os controladores de memória, reduzindo o impacto da contenção no acesso à memória.

Ribeiro [Ribeiro 2011] propôs Minas, uma infraestrutura para gerenciamento de afinidade de memória para arquiteturas ccNUMA. Em seus experimentos, a autora compara o desempenho fornecido pela interface proposta MAi (*Memory Affinity Interface*) com as políticas *first-touch* e *Interleave*. Seus resultados mostraram que em aplicações com padrão acesso aleatório ambas as políticas de alocação podem fornecer melhor desempenho que a distribuição manual dos dados.

As técnicas de migração manual possuem dois problemas. Primeiro, o programador deve conhecer o padrão de acesso à memória da aplicação e inserir chamadas ao sistema operacional para redistribuir os dados. Segundo, o custo de migração é alto se comparado à cópia de dados entre nós, pois envolve mecanismos de sincronização e atualização das tabelas de endereçamento [Goglin and Furmento 2009].

Outros trabalhos propõem a migração automática de páginas utilizando chamadas ao sistema operacional, contudo, os critérios para detecção e migração são diferentes. Nikoloupolos *et al.* [Nikolopoulos et al. 2000] propõem a modificação do compilador para instrumentação de regiões de memórias quentes, para durante a execução serem monitoradas e possivelmente migradas. Nesta técnica, o objetivo é reduzir a quantidade de memória que ser monitorada durante a execução.

Tikir e Hollingsworth [Tikir and Hollingsworth 2004] propõem o uso do *plugin* de *hardware* Sun Fire Link para monitorar os acessos remotos à memória das aplicações e com isto decidir sobre a necessidade da migração. Já Verghese *et al.* [Verghese et al. 1996] melhoram o desempenho de aplicações utilizando a migração e replicação dinâmica de páginas do sistema operacional SGI IRIX. Já Marathe *et*

al. [Marathe et al. 2010] propõem o perfilamento da versão incompleta das aplicações para obtenção de estatísticas sobre padrão de acesso à memória e com estas informações definir afinidade entre dados e threads para a execução completa.

Piccoli *et al.* [Piccoli et al. 2014] propõem técnicas de compilação para instrumentar aplicações para detecção e migração de páginas. Os autores utilizam um preditor de tamanho dos laços para ativar a migração somente em situações onde os benefícios são maiores do que a sobrecarga envolvida com o processo.

Broquedis *et al.* [Broquedis et al. 2010] propuseram o sistema ForestGOMP, que combina um escalonador de *threads* com um gerenciador de memória para arquiteturas ccNUMA. Da mesma forma que outros trabalhos, este sistema também utiliza chamadas ao sistema operacional para migrar páginas.

O mecanismo automático de balanceamento NUMA do Linux, da mesma forma que os outros de migração automática, migra as páginas e/ou tarefas de forma transparente, tentando reduzir o número de acessos remotos. Contudo, o mesmo não utiliza modificações no compilador ou no código da aplicação. Em vez de utilizar os contadores de *hardware* para detectar os acessos remotos, o mecanismo do Linux faz uso da unidade de gerenciamento de memória para induzir faltas de páginas e monitorar os acessos através de amostragem. Esta abordagem acarreta uma sobrecarga maior e é menos precisa do que a leitura dos contadores de *hardware*, entretanto, é mais portável pois a unidade de gerenciamento de memória é uma unidade presente nos diferentes processadores para os quais o núcleo do Linux é compilado.

7. Conclusões

Neste trabalho nós apresentamos uma avaliação do desempenho do mecanismo automático de balanceamento NUMA do Linux e da política de alocação *Interleave* em um sistema ccNUMA.

Assim como em trabalhos anteriores, nossos resultados indicam que o desempenho de aplicações paralelas em sistemas ccNUMA pode ser muito sensível à localização dos dados e tarefas, chegando a ser até 5x mais rápido quando os dados são distribuídos de forma melhor no sistema. Além disso, nossos resultados indicam que tanto a política de alocação de memória *Interleave* quanto o mecanismo automático de balanceamento NUMA do Linux podem ser usados para melhorar a distribuição dos dados e, consequentemente, o desempenho de aplicações paralelas em sistemas ccNUMA.

Mostramos ainda que em diversas situações a política de alocação de memória *Interleave* provê resultados de desempenho melhores do que o mecanismo automático de balanceamento NUMA e, por fim, apresentamos resultados que indicam uma forte correlação entre o potencial de ganho de desempenho e a taxa de acessos à memória das aplicações.

Referências

Bailey, D., Barszcz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Fineberg, S., Frederickson, P., Lasinski, T., Schreiber, R., Simon, H., Venkatakrishnan, V., and Weeratunga, S. (1994). The nas parallel benchmarks. *RNR Technical Report RNR M*.

Borin, E., Devloo, P. R. B., Vieira, G. S., and Shauer, N. (2014). Accelerating engineering software on modern multi-core processors. *Advances in Engineering Software*.

Broquedis, F., Furmento, N., Goglin, B., Wacrenier, P.-A., and Namyst, R. (2010). ForestGOMP: an efficient OpenMP environment for NUMA architectures. *International Journal of Parallel Programming*, 38(5-6):418–439.

Goglin, B. and Furmento, N. (2009). Enabling high-performance memory migration for multithreaded applications on Linux. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*.

Gorman, M. (2012). Foundation for automatic NUMA balancing [LWN.net]. Acessado em 30 de março de 2015.

Majo, Z. and Gross, T. R. (2013). (Mis)understanding the NUMA memory system performance of multithreaded workloads. In *2013 IEEE International Symposium on Workload Characterization (IISWC)*, pages 11–22. IEEE.

Marathe, J., Thakkar, V., and Mueller, F. (2010). Feedback-directed page placement for cc-numa via hardware-generated memory traces. *Journal of Parallel and Distributed Computing*, 70(12):1204–1219.

Nikolopoulos, D. S., Papatheodorou, T. S., Polychronopoulos, C. D., Labarta, J., and Ayguadé, E. (2000). User-level dynamic page migration for multiprogrammed shared-memory multiprocessors. *Parallel Processing, 2000. Proceedings. 2000 International Conference on*, 95–103.

Nordén, M., Löf, H., Rantakokko, J., and Holmgren, S. (2008). Geographical locality and dynamic data migration for OpenMP implementations of adaptive PDE solvers. In *OpenMP Shared Memory Parallel Programming*, pages 382–393.

Piccoli, G., Santos, H. N., Rodrigues, R., Pousa, C., Borin, E., and Pereira, F. M. Q. (2014). Compiler support for selective page migration in NUMA architectures. In *PACT*.

Ribeiro, C. P. (2011). *Contributions on Memory Affinity Management for Hierarchical Shared Memory Multi-Core Platforms*. PhD thesis, University of Grenoble.

Tikir, M. M. and Hollingsworth, J. K. (2004). Using hardware counters to automatically improve memory performance. In *Supercomputing, 2004. Proceedings of the ACM/IEEE SC2004 Conference*.

Verghese, B., Devine, S., Gupta, A., and Rosenblum, M. (1996). Operating system support for improving data locality on CC-NUMA compute servers. *Technical Report*, 31(9).

Wittmann, M. and Hager, G. (2010). Optimizing ccNUMA locality for task-parallel execution under OpenMP and TBB on multicore-based systems. *arXiv preprint arXiv:1101.0093*.

Yang, R., Antony, J., and Rendell, A. (2009). Effective use of dynamic page migration on NUMA platforms: The Gaussian chemistry code on the Sunfire X4600M2 system. *Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on*, pages 63–68.