# Extensão de um elemento de artigo

**Gustavo Ciotto Pinton**[1]

[1]Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)
Av. Albert Einstein, 1251, Cidade Universitária, Campinas/SP
Brasil, CEP 13083-852, Fone: [19] 3521-5838

ra117136@unicamp.br

***Abstract.*** *This report extends the results obtained in report 3 and appends two new curves: one representing 4KB memory pages and the other, 4MB memory pages. In this case, according to the orientation given during class, all results were calculated (or recalculated) for the* `SPEC CPU2006` *benchmarks containing 100 million instructions in the warm up region and 30 million instructions in the detailed region. As in the previous case, we used micro-archicteture simulator sniper and its internal structure was modified in order to reflect the desired configuration. We discussed the effects of different page sizes on the measured IPC and considered a page table with a three-level hierarchy.*

***Resumo.*** *Este relatório estende os resultados obtidos no relatório 3 e acrescenta duas novas curvas: uma para páginas de memória com 4KB e a outra, para 4MB. Neste caso, de acordo com a orientação do professor durante as aulas, todos os resultados foram calculados (ou recalculados) para os pinballs com 100 milhões de instruções na região de warm-up e 30 milhões na região detalhada dos benchmarks do* `SPEC CPU2006`*. Assim como no caso anterior, o simulador de micro-arquiteturas sniper foi utilizado e sua estrutura interna foi modificada a fim de refletir os estados desejados. Nós discutimos os efeitos da alteração do tamanho das páginas no IPC medidos e consideramos uma hierarquia de três níveis para a tabela de páginas.*

## 1. Introdução

O artigo escolhido para os projetos 3 e 4 foi *Accelerating Decoupled Look-ahead via Weak Dependence Removal: A Metaheuristic Approach* [Parihar and Huang 2014]. Neste artigo, em poucas palavras, os autores propuseram uma maneira a partir de algoritmos genéticos de melhorar o desempenho da *thread* auxiliar (chamada também de *look-ahed thread*) que, segundo seus experimentos, torna-se o limitante da performance dos programas *single-threaded* que são executados neste tipo de arquitetura. Um resumo completo do artigo e do relatório 3 pode ser encontrado em [Pinton 2016b].

No projeto 3, implementamos as curvas *ideal* e *single-thread* da figura 1 (figura 3 da página 3 de [Parihar and Huang 2014]). Propõem-se, neste relatório, adicionar dois novos resultados a este gráfico, referentes a páginas de memória de 4KB e de 4MB. Diferentemente do projeto 2 [Pinton 2016a], em que o objetivo foi avaliar o número de acessos adicionais à memória devido a *misses* nas TLBs, o intuito deste relatório é avaliar os efeitos da mudança do tamanho das páginas no IPC. Assim como em [Pinton 2016b],

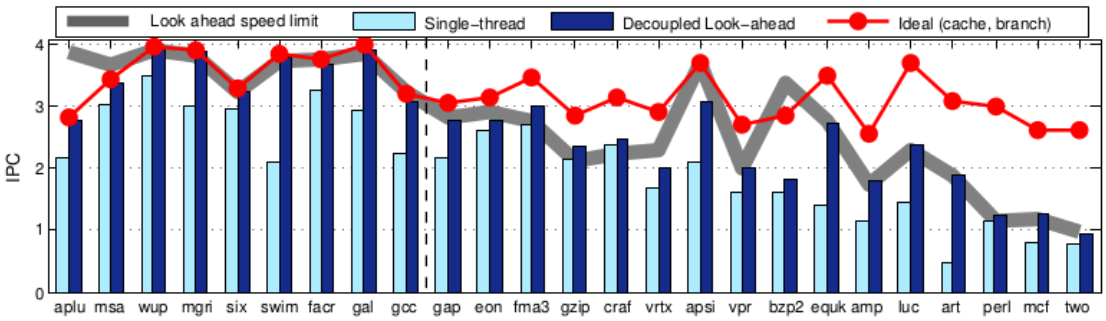


**Figure 3.** Performance comparison of 4 configurations. Shown in the bars are baseline single core (left) and a decoupled look-ahead system (right). Two upper-bounds are shown: the performance of a single core with idealized branch predictions and perfect cache accesses (curve with circles), and the the approximate speed limit of the look-ahead thread (gray wide curve indicating approximation). The applications are sorted with increasing performance gap between the decoupled look-ahead system and the prediction- and accesses-idealized single-core system.

**Figura 1. Figura escolhida para reprodução em [Pinton 2016b] com legenda original. Extraída de [Parihar and Huang 2014].**

utilizamos o *sniper* como simulador e modificamos algumas de suas classes para permitir a especificação do tamanho de tais páginas. Essas modificações serão detalhadas nas próximas seções.

No projeto 2 [Pinton 2016a], concluimos que o número de acessos a memória era superior para páginas de 4KB. Adicionalmente, para alguns *benchmarks* testados nesta ocasião, o número de acessos ocasionadas por *misses* na TLB correspondia a uma fração importante do número total de acessos. Entretanto, discutimos também que, apesar disso, o desempenho no caso de 4MB não era necessariamente melhor que para 4KB, visto que páginas de 4MB introduziriam uma forte fragmentação da memória já que até processos utilizando menos de 4MB teríam uma página completa alocada na memória.

A configuração do ambiente utilizado neste relatório seguiu aquela especificada em [Pinton 2016b]. Entretanto, ao invés de usar *pinballs* de *benchmarks* do `SPEC CPU2006` com 1 bilhão de instruções na região detalhada e sem *warm up*, utilizamos aqueles com 100 milhões de intruções de aquecimento e 30 milhões para a região de interesse. Estes últimos oferecem resultados mais próximos dos originais, uma vez que eliminam os efeitos dos *misses* e *mispredictions* iniciais. Além disso, recalculamos os resultados obtidos em [Pinton 2016b] para tais *pinballs*.

Consideramos para este relatório que a tabela de páginas contém 3 níveis de hierarquia. Dessa forma, a fim de simular esse comportamento, foi necessário o estudo do código do *sniper* e a identificação dos parâmetros que pudessem adicionar a latência de 3 acessos adicionais a memória em caso de um eventual *miss* nas TLBs de instruções e dados. As modificações realizadas podem ser encontradas na seção **Implementação**, a seguir.

## 2. Implementação

Nesta seção, serão discutidos detalhes da implementação, tais como modificações em classes do *sniper* e na sua configuração. Destaca-se, conforme já mencionado na seção anterior, que estendemos a configuração apresentada no relatório do projeto 3 [Pinton 2016b] para incluir páginas de memória de diferentes tamanhos.

### 2.1. Modificação do *sniper*

As TLBs são implementados pelo simulador *sniper* através da classe `TLB`, declarada no arquivo `tlb.h` e definida em `tlb.cc`. Originalmente, essa classe possui três contantes, sendo elas `SIM_PAGE_SHIFT`, `SIM_PAGE_SIZE` e `SIM_PAGE_MASK`, representando, respectivamente, o deslocamento em bits que deve ser efetuado no endereço a ser traduzido, o tamanho de uma pagína de memória e a máscara a ser utilizada. É importante observar que `SIM_PAGE_SIZE` e `SIM_PAGE_MASK` são definidos por um único parâmetro, sendo ele `SIM_PAGES_SHIFT`. Dessa maneira, qualquer modificação neste último, altera todo o comportamento na classe. Por padrão, o *sniper* define o valor de `SIM_PAGE_SHIFT` igual a 12, utilizando assim páginas de 4KB (4KB = $2^{12}$). Para o uso de páginas de 4MB, por sua vez, `SIM_PAGE_SHIFT` deve valer 22 (4MB = $2^{22}$).

Modificamos, portanto, o *sniper* e transformamos as constantes do parágrafo em variáveis. Adicionamos ao construtor da classe um novo parâmetro, chamado de `page_size_bits`, que determina o valor de `SIM_PAGE_SHIFT` e, em consequência, de `SIM_PAGE_SIZE` e `SIM_PAGE_MASK`. A determinação deste novo parâmetro é realizada a partir da opção `page_size_bits`, lida dos arquivos de extensão `.cfg`.

Alteramos o construtor da classe `MemoryManager`, implementado no arquivo `memory_manager.cc`, para a leitura desta nova opção e recompilamos o *sniper*. O arquivo de configuração `base.cfg` que serve como base para todas as configurações define a opção `page_size_bits` como 12. Assim, configurações criadas anteriormente às modificações desta seção continuarão a utilizar as páginas padrões de 4KB.

Enfim, atribuímos à propriedade `penalty` de `perf_model/tlb` o valor de 600 ($3 \times 200$), especificando assim a penalidade de três acessos adicionais à memória em caso de um *miss* em qualquer das TLBs.

### 2.2. Correção na propriedade *latency* da memória

No relatório 3 [Pinton 2016b], citamos que o a latência de memória correspondia a 200 ciclos. Nesta ocasião, atribuímos à propriedade `latency` de `perf_model/dram` o valor de 200. Entretanto, o *sniper* interpreta este valor como *nanossegundos* e não ciclos. Dessa forma, a latência especificada corresponde a um valor muito superior daquela desejada, isto é, 87ns (para a frequência de 2.3GHz, 200 ciclos equivalem a $200 * \frac{1}{2.3 \times 10^9} \approx 87ns$). Portanto, os valores de IPC encontrados em [Pinton 2016b] foram um pouco inferiores aos desejados. Entretanto, como este parâmetro foi usado para todos os *benchmarks*, a forma das curvas encontrada deve se manter igual àquela encontrada neste relatóriom na seção **Resultados**.

### 2.3. Estrutura das *caches*

Escolhemos utilizar a estrutura de caches implementado na configuração da microarquitetura *Nehalem*, presente por padrão no *sniper*. Dois níveis de *cache* estão presentes, sendo que o segundo é compartilhado entre as TLBs de dados e instruções.

A TLB de instruções possui 128 entradas e é *4-way associative*. A de dados, por sua vez, possui a metade de entradas, isto é, 64, e também é *4-way associative*. Por fim, a TLB compartilhada possui 512 entradas e, como as duas anteriores, é *4-way associative*.

## 3. Resultados

A figura 3 representa os resultados obtidos para três tipos de processamento: o ideal, em que *mispredictions* e *cache misses* são desconsiderados, e aqueles que utilizam páginas de memória de 4KB e 4MB. O gráfico está disposto em ordem crescente de diferença entre a curva ideal e o resultado obtido para o caso de 4MB. Destaca-se que o IPC medido para tal comprimento de página para a entrada `ref 1` do *benchmark* `hmmer` é o que mais se aproxima do seu desempenho ideal. Observa-se, ainda, que, para este *benchmark*, a diferença entre os resultados obtidos é o menor entre todos os *benchmarks*, conforme figura 3. Uma possível explicação para tal comportamento reside no fato de que este *benchmark* apresenta um dos menores *footprints* de memória entre todos aqueles que compõem o `CPU 2006` [Henning 2007]. Dessa forma, como não há muitas operações de memória e, portanto, *misses* nas TLBs, a diferença provocada no IPC pela mudança do tamanho das páginas não é significativa. Em outras palavras, a mudança de 4KB para 4MB não produzirá uma diminuição importante nos acessos à memória causados por leitura à tabela de páginas. Por outro lado, os IPCs medidos para o `milc` foram os que mais se distanciaram da respectiva curva ideal. Assim como em `hmmer`, `milc` apresenta um *footprint* de memória bem característico: tal *benchmark* contém uma das maiores utilizações de memória entre todo o conjunto [Henning 2007]. Sendo assim, as penalizações causadas por *cache misses* e acessos à tabela de páginas produzem degradações mais importantes nos IPCs medidos. De maneira geral, ainda apoiando-se neste raciocínio, é possível estender este resultado para toda a figura 3: *benchmarks* com menor *footprint* de memória apresentam medidas mais próximas do ideal e vice-versa.

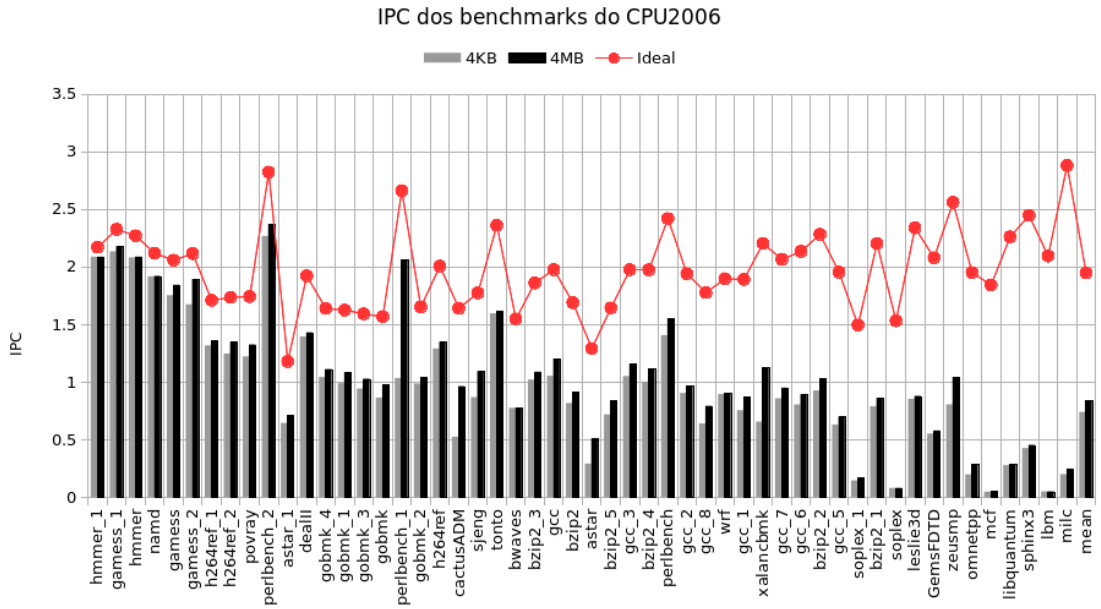


**Figura 2. IPCs medidos para *benchmarks* do `SPEC CPU2006`.**

A figura 3 apresenta, em ordem crescente, as diferenças entre os processamentos com páginas de 4MB e 4KB para todos os *benchmarks*. Obteve-se uma diferença média de apenas 0.11 no IPC, indicando, assim, que o tamanho da página de memória não é um fator determinante no aumento do IPC, contrariamente a outras estruturas, tais como

o número de portas de leitura e escrita da *issue queue*, por exemplo, conforme vimos em aula. A maior diferença encontrada foi para a entrada `ref 1` de `perlbench`. Assim como o *toy benchmark* desenvolvido em [Pinton 2016a], tal entrada é muito mais adaptada para páginas de 4MB: muito provavelmente, ela produz muitas trocas nos blocos da TLB de dados para páginas de 4KB e que não ocorrem para 4MB.

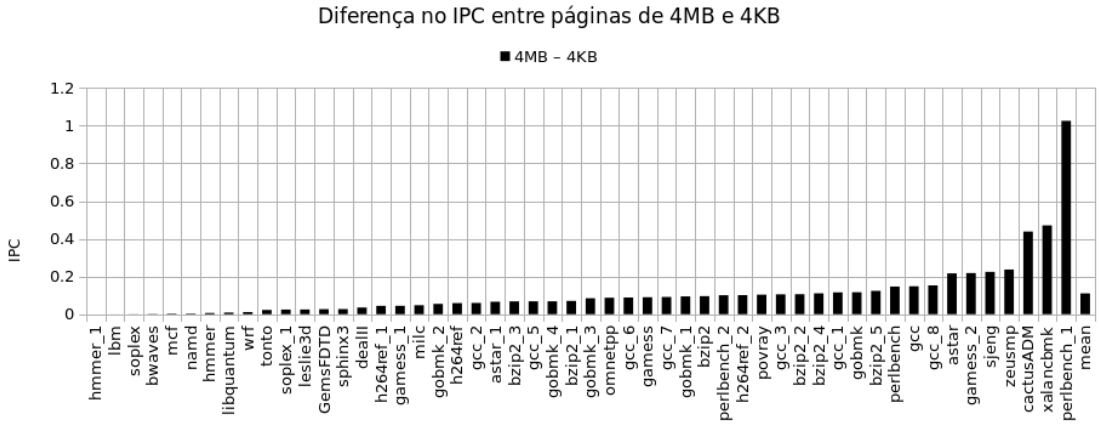


**Figura 3. Diferença entre os IPCs medidos para 4KB e 4MB nos *benchmarks* do `SPEC CPU2006`.**

As figuras 4 e 5 apresentam, respectivamente, as diferenças entre os resultados obtidos neste relatório e no 3 [Pinton 2016b] para as curvas ideias e para páginas de 4KB, respectivamente. No projeto 3, utilizaram-se *pinballs* com 1 bilhão de instruções a região detalhada e nenhuma para *warm up*, enquanto que, no 4, usamos aqueles referentes a 100 milhões de intruções de aquecimento e 30 milhões na região de interesse. Observa-se que o comportamento para as curvas ideais é muito semelhante entre as duas situações, já que as mesmas penalidades foram retiradas de ambos os processamentos. Ligeiras variações entre as curvas da figura 4 são obtidas apenas a partir do *benchmark* `milc`.

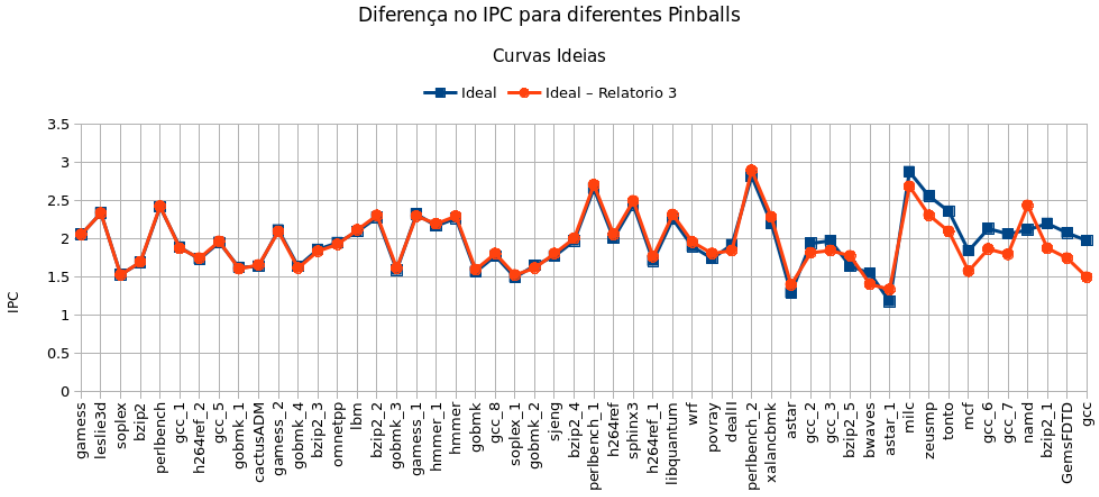


**Figura 4. Diferença entre as curvas ideais obtidas a partir de dois tipos de *pinballs*.**

Para os processamentos *reais*, considerando *misses* e *mispredictions*, entretanto, as diferenças entre os resultados obtidos são mais aparentes, conforme mostra a figura 5. Observamos divergências consideráveis na medida de IPC para a maior parte de *benchmarks*, sendo que não há um comportamento bem definido: para alguns deles o IPC do relatório 3 foi superior àquele do 4 e vice-versa.

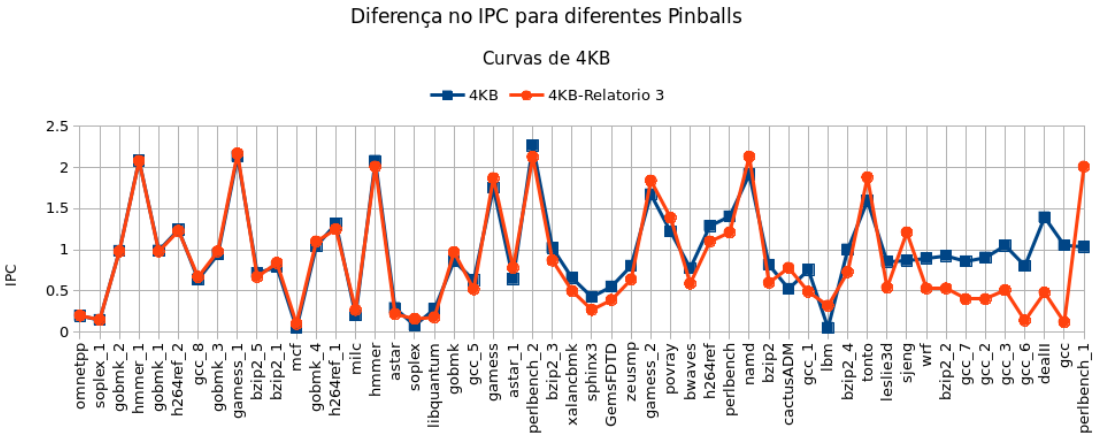


**Figura 5. Diferença entre as curvas de 4KB obtidas a partir de dois tipos de *pinballs*.**

## 4. Conclusões

Foi possível estendermos os resultados obtidos no relatório 3 [Pinton 2016b] para páginas de 4MB através da configuração de alguns aspectos do simulador *sniper*. Discutimos e relacionamos os resultados encontrados com os *footprints* de memória dos *benchmarks*, documentados em [Henning 2007]. Concluímos que a modificação do tamanho das páginas da memória não é um fator determinante para o IPC e obtivemos um ganho, em média, de 0.11 no IPC, para páginas de 4MB. Entretanto, não consideramos efeitos como fragmentação de memória, que poderiam degradar tais medidas. Além disso, comparou-se os resultados obtidos a partir de *pinballs* contendo 1 bilhão de intruções sem *warm up* em [Pinton 2016b] com aqueles contendo 100 milhões de *warm up* e 30 milhões na região detalhada. Nenhuma grande divergência foi encontrada nas medidas do IPC entre estes dois casos.

## Referências


Henning, J. L. (2007). Spec cpu2006 memory footprint. *ACM SIGARCH Computer Architecture News*, 35.

Parihar, R. and Huang, M. C. (2014). Accelerating decoupled look-ahead via weak dependence removal: A metaheuristic approach. *International Symposium on High Performance Computer Architecture*.

Pinton, G. C. (2016a). Comparação do número de acessos à memória e tlb misses com páginas de 4kb ou 4mb. Relatório 2 apresentado na disciplina MO601.

Pinton, G. C. (2016b). Reprodução de um elemento de artigo. Relatório 3 apresentado na disciplina MO601.