



INSTITUTO SUPERIOR TÉCNICO

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

ORGANIZAÇÃO DE COMPUTADORES

LEIC

Primeira tarefa de laboratório: modelagem e perfil de sistema

Versão 1.1.0

2023/2024

1. Introdução

O objetivo deste trabalho é duplo: (i) determinar as características dos caches de um computador, e (ii) aproveitar o conhecimento obtido sobre os caches para otimizar o desempenho de um determinado programa. Para esta tarefa, os alunos farão uso de uma ferramenta de análise de desempenho para ter acesso direto aos contadores de desempenho de hardware disponíveis na maioria dos microprocessadores modernos. A ferramenta que será utilizada é a Application Programming Interface (API) padrão: PAPI [1].

No restante desta seção, fazemos uma breve introdução ao PAPI e descrevemos a plataforma computacional alvo e o ambiente de desenvolvimento. Na Seção 3, descrevemos o procedimento para modelar os caches L1 e L2 da plataforma alvo (Subseção 3.1) e fornecemos um guia para analisar o desempenho de um segmento de código de multiplicação de matrizes e otimizá-lo com base nas características do L2. cache da arquitetura alvo (Subseção 3.2).

1.1 Plataforma Alvo e Ambiente de Desenvolvimento

IMPORTANTE: Este trabalho deverá ser realizado nos computadores da sala de aula do seu laboratório.

Esses computadores possuem características de hardware semelhantes e qualquer um deles pode ser usado como plataforma de destino. Observe que, como este trabalho depende do hardware, realizá-lo em um computador com características de hardware diferentes pode produzir resultados inesperados e, portanto, invalidar o seu trabalho. Isso significa que você deve sempre usar o mesmo laboratório. Se for aluno da Alameda, poderá aceder ao computador específico do laboratório que desejar (ver <https://welcome.rnl.tecnico.ulisboa.pt/#labs-access>).

Para configurar corretamente o ambiente de desenvolvimento, é necessário obter a biblioteca PAPI e um conjunto de arquivos auxiliares de programas. Esse material pode ser encontrado no pacote `lab1_kit.zip`, que pode ser baixado no site do curso. Depois de baixar e descompactar este pacote em qualquer um dos computadores das turmas do laboratório, o PAPI deve ser construído. Para isso, altere os diretórios para o local do código-fonte do PAPI: pasta `papi-XXX/src`. Compile o código emitindo os comandos: `./configure` e `make`. Esta operação produzirá um conjunto de ferramentas auxiliares localizadas no diretório `src/utils/` e criará a biblioteca PAPI `papilib.a`. A ferramenta `papi_avail`, em particular, é útil para determinar os eventos PAPI suportados na plataforma de destino. A biblioteca estará vinculada aos programas auxiliares apresentados nas seções seguintes.

2 Exercício

Para ajudar a determinar as características dos caches dos computadores do laboratório, os exercícios a seguir ajudarão você a estimar os parâmetros de cache de pequenos aplicativos C.

O primeiro passo para se familiarizar com o procedimento é determinar apenas o tamanho do cache usando uma pequena aplicação C em uma máquina (conhecida), como o código que você analisou no exercício de laboratório VI.3.

Este código C é uma versão simplificada dos seguintes programas nesta tarefa. Basicamente, ele itera em um array para determinar o tamanho do cache.

Para garantir que você mede o tempo com precisão, use o código-fonte disponível no kit de laboratório (arquivo spark.c).

Para realizar a avaliação você deve ir ao seu laboratório para acessar o tamanho do cache executando a aplicação lá. Você pode repetir a avaliação do tempo decorrido algumas vezes para obter significância estatística. Você deve tabelar os resultados relevantes para diferentes tamanhos de cache na folha de respostas e tirar uma conclusão sobre o tamanho do cache. Você pode calcular mais medidas antes da saída, examine a parte final do arquivo de código-fonte.

1. Qual é a capacidade de cache do computador que você testou? Por favor, justifique.

Para descobrir os outros parâmetros de cache, você modificará a aplicação C, para que ela gere diferentes padrões de acesso a dados. Por favor, gaste alguns minutos analisando as modificações no código-fonte.

```
for(tamanho_t tamanho_cache = CACHE_MIN; tamanho_cache < CACHE_MAX; tamanho_cache = 2*tamanho_cache) {
    for(size_t passada = 1; passada <= cache_size/2; passada = 2*passada){
        limite = cache_size - passada + 1; for(ssize_t i = 10
            * passada; i > 0; i--) {
            for(índice = 0; índice <limite; índice += passada) { array[índice] =
                array[índice] + 1;
            }
        }
    }
}
```

O significado de cada variável é o seguinte:

array[] um grande array arbitrário que será acessado repetidamente para medir o padrão de falta de cache;

valor cache_size do tamanho do cache em teste; todos os tamanhos de cache dados por potências inteiras de 2, entre CACHE_MIN = 8kB e CACHE_MAX = 64kB devem ser considerados;

stride informa quantas entradas estão sendo ignoradas em cada acesso; por exemplo, se a passada for 4, as entradas 0, 4, 8, 12, ... na matriz estão sendo acessadas, enquanto as entradas 1, 2, 3, 5, 6, 7, 9, 10, 11, .. são ignorados;

limitar o maior endereço que será acessado para o tamanho do cache e padrão de acesso em teste;

repetir denota o número de vezes que cada padrão de acesso será repetido no array.

O tempo de execução deste segmento de código nesta máquina resulta no gráfico mostrado na Figura 1, variando o valor adotado para o parâmetro stride e para diferentes tamanhos de array, definidos entre ARRAY_MIN = 4kB e ARRAY_MAX = 4MB.

2. Qual é a capacidade de cache do computador?
3. Qual é o tamanho de cada bloco de cache?
4. Qual é o tempo de penalidade por falta de cache L1?

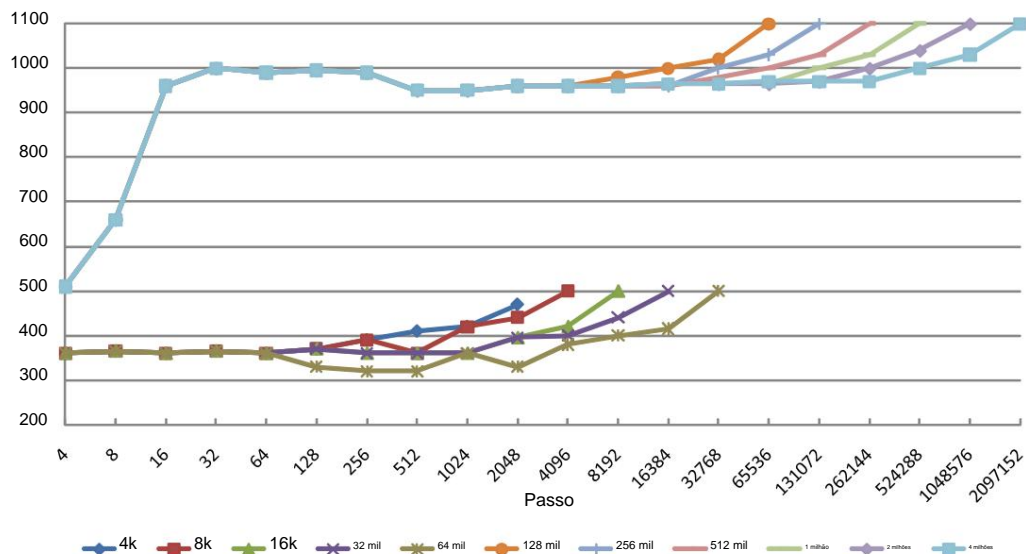


Figura 1: Variação do tempo de acesso ao cache com o valor de stride adotado para diferentes tamanhos de array.

3 Procedimento

3.1 Modelando Caches de Computador

Na primeira parte deste trabalho, o objetivo é modelar as características do cache de dados L1 e do cache L2 da plataforma computacional alvo. A seguir, fornecemos instruções para realizar esta análise.

Utilize os formulários ao final para responder às questões abaixo.

3.1.1 Modelando o Cache de Dados L1

A metodologia para modelar experimentalmente o cache de dados L1 consiste em considerar a quantidade total de perdas de cache de dados durante a execução da seguinte sequência de código do programa cm1.c, semelhante ao programa da Seção 2. Este programa pode ser encontrado no pacote lab1_kit .fecho eclair.

```
for(array_size=ARRAY_MIN; array_size < ARRAY_MAX; array_size=array_size*2) for(stride=1; stride <=
array_size/2; stride=stride*2){ limit = array_size - stride + 1; for(repetir=0;
repetir<=200*passo; repetir++)

    for(índice=0; índice<limite; índice+=passo) x[índice] = x[índice] +
        1;
}
```

a) Vá para o diretório cm1/, no pacote lab1_kit.zip, e analise o código do programa

cm1.c. Identifique seu código-fonte com o programa descrito acima.

Quais são os eventos do processador que serão analisados durante sua execução? Explique seu significado.

b) Compile o programa cm1.c usando o Makefile fornecido e execute cm1. Trace a variação do número médio de erros (Avg Misses) com o tamanho da passada, para cada dimensão considerada do cache de dados L1 (8kB, 16kB, 32kB e 64kB).

NOTA: Um esboço rápido desses gráficos pode ser desenhado em seu computador executando os seguintes comandos:

```
./cm1 > cm1.out
```

```
./cm1_proc.sh
```

2: Você pode desenhar essas tabelas e gráficos em seu computador, imprimir e anexar ao relatório. Você não tem que

preencha-os à mão no relatório impresso.

NOTA 3: Pode ser necessário marcar o script como executável antes de poder executá-lo.

c) Analisando os resultados obtidos:

- Determine o tamanho do cache de dados L1. Justifique sua resposta. _____
- Determine o tamanho do bloco adotado neste cache. Justifique sua resposta. _____
- Caracterizar o tamanho do conjunto de associatividade adotado nesta cache. Justifique sua resposta. _____

3.1.2 Modelando o Cache L2

Nesta parte do trabalho, o objetivo é modelar experimentalmente as características do cache L2 da plataforma computacional alvo. Para analisar o cache L2 do computador, usaremos a mesma metodologia apresentada na seção anterior para modelar o cache de dados L1.

a) Modifique o programa cm1.c para analisar as características do cache L2. (Dica: utilize o evento PAPI_L2_DCM.) Descreva e justifique as alterações introduzidas neste programa. _____

b) Compile o programa cm1.c, execute cm1 e plote a variação do número médio de erros (Avg Misses) com o tamanho da passada, para cada dimensão considerada do cache L2.

c) Analisando os resultados obtidos:

- Determine o tamanho do cache L2. Justifique sua resposta. _____
- Determine o tamanho do bloco adotado neste cache. Justifique sua resposta. _____
- Caracterizar o tamanho do conjunto de associatividade adotado nesta cache. Justifique sua resposta. _____

3.2 Criação de perfil e otimização de acessos ao cache de dados

Freqüentemente, os programadores que desejam melhorar o desempenho de seus programas concentram sua atenção em como os programas afetam os caches do computador. A seguir, será analisado como simples mudanças de código podem ajudar a melhorar esse desempenho para uma aplicação de multiplicação de matrizes.

Considere uma aplicação simples de multiplicação de matrizes, operando em duas matrizes quadradas de $N \times N$ elementos inteiros de 16 bits, com $N = 1024$. Do ponto de vista matemático, dadas duas matrizes A e B , com elementos a_{ij} e b_{ij} tais que $0 \leq i, j < N$, a matriz de produto C é definida como:

$$c_{ij} = \sum_{k=0}^{N-1} a_{ik}b_{kj} = a_{i1}b_{1j} + a_{i2}b_{2j} + \dots + a_{i(N-1)}b_{(N-1)j} \quad (1)$$



Figura 2: Multiplicação direta de matrizes.

3.2.1 Implementação direta

Uma implementação C direta da Eq. 1 pode ficar assim:

```
para (eu = 0; eu < N; ++i) {
    para (j = 0; j < N; ++j) {
        para (k = 0; k < N; ++k) {
            res[i][j] += mul1[i][k] * mul2[k][j];
        }
    }
}
```

As duas matrizes de entrada são mul1 e mul2. A matriz de resultados res é assumida como inicializada para todos zeros.

O programa fornecido mm1.c inclui esta sequência de código e todas as etapas de inicialização necessárias, bem como o conjunto de instruções necessárias para traçar o perfil de sua execução usando a caixa de ferramentas PAPI.

a) Mude para o diretório mm1/ e analise o código do programa mm1.c. Identifique seu código-fonte com o programa descrito acima.

Qual é a quantidade total de memória necessária para acomodar cada uma dessas matrizes?

b) Compile o arquivo fonte mm1.c usando o Makefile fornecido e execute-o. Preencha a tabela com os dados obtidos.

c) Avalie a taxa de acerto do cache de dados L1 resultante.

3.2.2 Primeira Otimização: Transposição de matriz antes da multiplicação [2]

Ao analisar os resultados obtidos, pode-se observar que uma implementação tão simples sofre uma penalidade severa no que diz respeito à quantidade de perdas de cache L2 resultantes do seu padrão de acesso.

Na verdade, enquanto a matriz mul1 é acessada sequencialmente, o loop interno avança o número da linha de mul2 (ver Fig. 2), significando acessos sucessivos a posições de memória distantes.

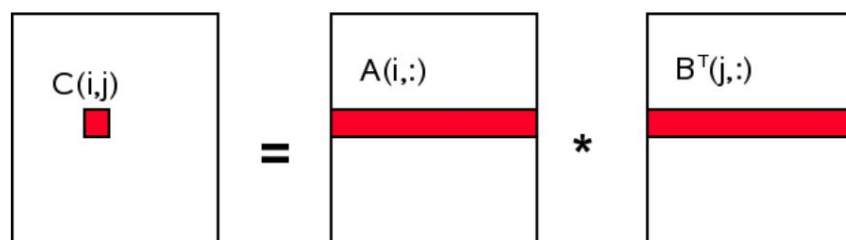


Figura 3: Multiplicação de matrizes transpostas.

Uma possível solução para atenuar tal problema baseia-se na transposição de matrizes. Na verdade, como cada elemento da matriz é acessado múltiplas vezes, pode valer a pena reorganizar ("transpor", em termos matemáticos) a segunda matriz mul2 antes de usá-la (ver Fig. 3):

$$c_{ij} = \sum_{k=0}^{N-1} a_{ik} b_{kj} = a_{i1} b_{j1} + a_{i2} b_{j2} + \dots + a_{i(N-1)} b_{j(N-1)} \quad (2)$$

Após a etapa preliminar de transposição, ambas as matrizes podem ser iteradas sequencialmente. Tanto quanto o C código está em questão, agora ele se parece com isto:

```
int16_t tmp[N][N];

//transposição para (i =
0; i < N; ++i) {
    para (j = 0; j < N; ++j) {
        tmp[i][j] = mul2[j][i];
    }
}

// multiplicação para (i =
0; i < N; ++i) {
    para (j = 0; j < N; ++j) {
        para (k = 0; k < N; ++k) { res[i][j] +=
mul1[i][k] * tmp[j][k]; }
    }
}
```

A variável tmp é um array temporário para armazenar a matriz transposta.

Uma consequência direta desta otimização é que ela agora requer acessos adicionais à memória de dados. Esperançosamente, esse custo extra pode ser facilmente recuperado, já que os 1.024 acessos não sequenciais por coluna costumam ser bem mais caros.

- a) Mude para o diretório mm2/ e analise o código do programa mm2.c. Identifique seu código-fonte com o programa descrito acima. Compile este programa usando o Makefile fornecido e execute-o.

Preencha a tabela com os dados obtidos.

- b) Avalie a taxa de acerto do cache de dados L1 resultante.

- c) Altere o código no programa mm2.c para incluir a transposição da matriz no tempo de execução. Compile este programa usando o Makefile fornecido e execute-o.

Preencha a tabela com os dados obtidos.

Comente os resultados obtidos ao incluir a transposição da matriz no tempo de execução.

- d) Comparar os resultados obtidos com os obtidos para a implementação simples, calculando a diferença entre as taxas de acerto resultantes (γHitRate) e os speedups obtidos.

3.2.3 Segunda Otimização: Multiplicação de matriz bloqueada (ladrilhado) [2]

Apesar dos bons resultados que podem ser obtidos com o método de transposição de matrizes, em muitas aplicações esta abordagem não pode ser aplicada, seja porque a matriz é muito grande ou porque a memória disponível é muito pequena. Assim, outras alternativas, que não exijam o procedimento de cópia extra, devem ser estudadas.

A busca por um esquema de processamento alternativo deve começar com um exame minucioso da matemática envolvida e das operações realizadas pela implementação original. O conhecimento trivial da matemática mostra que a ordem das diversas adições para obter cada elemento da matriz de resultados é irrelevante, desde que

cada adendo aparece exatamente uma vez. Esse entendimento levará a soluções que reordenam as adições realizadas no loop interno do código original.

Segundo o algoritmo original, a ordem adotada para acessar os elementos da matriz $mul2$ é: $(0,0), (1,0), \dots, (N-1,0), (0,1), (1,1), \dots$. Embora os elementos $(0,0)$ e $(0,1)$ estejam na mesma linha de cache, no momento em que o loop interno completa uma rodada, essa linha de cache já foi removida há muito tempo. Para este exemplo, cada rodada do loop interno requer, para cada uma das três matrizes, 1.024 linhas de cache, o que é muito mais do que está disponível nos caches da maioria dos processadores.

Uma solução possível é lidar simultaneamente com mais de uma iteração do loop intermediário, enquanto executa o loop interno. Neste caso, serão utilizados vários valores que têm garantia de estar em cache, contribuindo assim para uma redução da taxa de erros do cache L2. Assim, para maximizar o speedup proporcionado por esta técnica, é necessário adaptar a dimensão da submatriz em processamento ao tamanho do bloco de cache, tendo em conta o tamanho de cada elemento da matriz. Como exemplo hipotético, considerando que um operando curto ocupa 2 bytes, isso significa que um bloco de cache de 64 bytes acomodará 32 elementos da matriz, definindo assim o tamanho ideal para a linha da submatriz como 32 (ver Fig. 4).

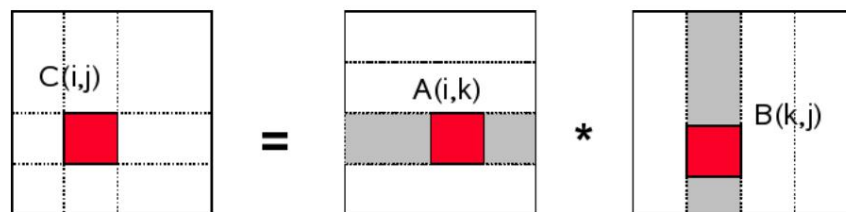


Figura 4: Multiplicação de matrizes bloqueadas.

No que diz respeito ao código C, agora fica assim:

```
#define SUB_MATRIX_SIZE (CACHE_LINE_SIZE / sizeof (curto))

para (i = 0; i < N; i += SUB_MATRIX_SIZE) {
    para (j = 0; j < N; j += SUB_MATRIX_SIZE) {
        para (k = 0; k < N; k += SUB_MATRIX_SIZE) { para (i2 = 0, rres
            = &res[i][j], rmul1 = &mul1[i][k];
                i2 < SUB_MATRIX_SIZE;
                ++i2, rres += N, rmul1 += N) {
                    para (k2 = 0, rmul2 = &mul2[k][j]; k2 < SUB_MATRIX_SIZE; ++k2, rmul2 += N) {
                        para (j2 = 0; j2 < SUB_MATRIX_SIZE; ++j2) { rres[j2] += rmul1[k2]
                            * rmul2[j2];
                        }
                    }
                }
            }
        }
    }
}
```

A mudança mais visível é que o código agora possui seis loops aninhados. Os loops externos iteram com intervalos de SUB_MATRIX_SIZE (o tamanho da linha de cache CACHE_LINE_SIZE dividido por sizeof(short)).

Isso divide a multiplicação de matrizes em vários problemas menores que podem ser resolvidos com mais localidade de cache. Os loops internos iteram sobre os índices ausentes dos loops externos. Existem, mais uma vez, três loops. Os loops $k2$ e $j2$ estão em uma ordem diferente. Isto é feito porque, no cálculo real, apenas uma expressão depende de $k2$, mas duas dependem de $j2$.

- a) Mude para o diretório mm3/ e analise o código do programa mm3.c. Identifique seu código-fonte com o programa descrito acima.

Altere o código fonte do programa para adequar a parametrização do algoritmo (tamanho da linha da submatriz) ao tamanho do bloco (CLS) que foi determinado na Seção 3.1.

Quantos elementos da matriz podem ser acomodados em cada linha de cache?

- b) Compile este programa usando o Makefile fornecido e execute-o. Preencha a tabela com o obtido dados.

- c) Avalie a taxa de acerto do cache de dados L1 resultante.
- d) Comparar os resultados obtidos com os obtidos para a implementação simples, calculando a diferença entre as taxas de acerto resultantes (γ HitRate) e o speedup obtido.
- e) Comparar os resultados obtidos com os obtidos para a implementação da transposição de matrizes calculando a diferença entre as taxas de acerto resultantes (γ HitRate) e o speedup obtido.
Se a aceleração obtida for positiva, mas a diferença nas taxas de acerto resultantes for negativa, como você explica a melhoria de desempenho? (Dica: estude as taxas de acerto do cache L2 para ambas as implementações; você pode usar os seguintes eventos PAPI PAPI_L2_DCH (ou PAPI_L2_DCM) e PAPI_L2_DCA. Execute papi_avail para verificar os eventos disponíveis e entender seu significado.)

Referências

- [1] Interface de programação de aplicativos de desempenho (PAPI). Página da Internet. "<http://icl.cs.utk.edu/papi>", dezembro de 2008.
- [2] Ulrich Drepper. O que todo programador deveria saber sobre memória. Relatório técnico, Red Hat, Inc., novembro de 2007.
- [3] Guia do usuário PAPI.
- [4] Referência do programador PAPI.

Primeira tarefa de laboratório: modelagem e perfil de sistema

IDENTIFICAÇÃO DOS ESTUDANTES:

Número:	Nome:

2 Exercício

Justifique todas as suas respostas com valores dos experimentos.

1. Qual é a capacidade de cache do computador que você usou (escreva o nome da estação de trabalho)?

Tamanho da matriz						
t2-t1						
# acessa a[i]						
# tempo médio de acesso						

--

Considere os dados apresentados na Figura 1. Responda às seguintes questões (2, 3, 4) sobre a máquina usada para gerar esses dados.

2. Qual é a capacidade do cache?

--

3. Qual é o tamanho de cada bloco de cache?

--

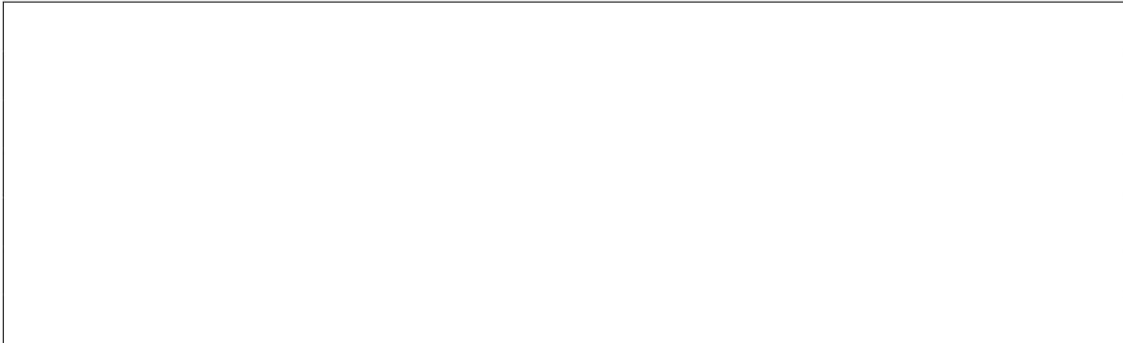
4. Qual é o tempo de penalidade por falta de cache L1?

--

3 Procedimento

3.1.1 Modelando o Cache de Dados L1

- a) Quais são os eventos do processador que serão analisados durante sua execução? Explique seu significado.

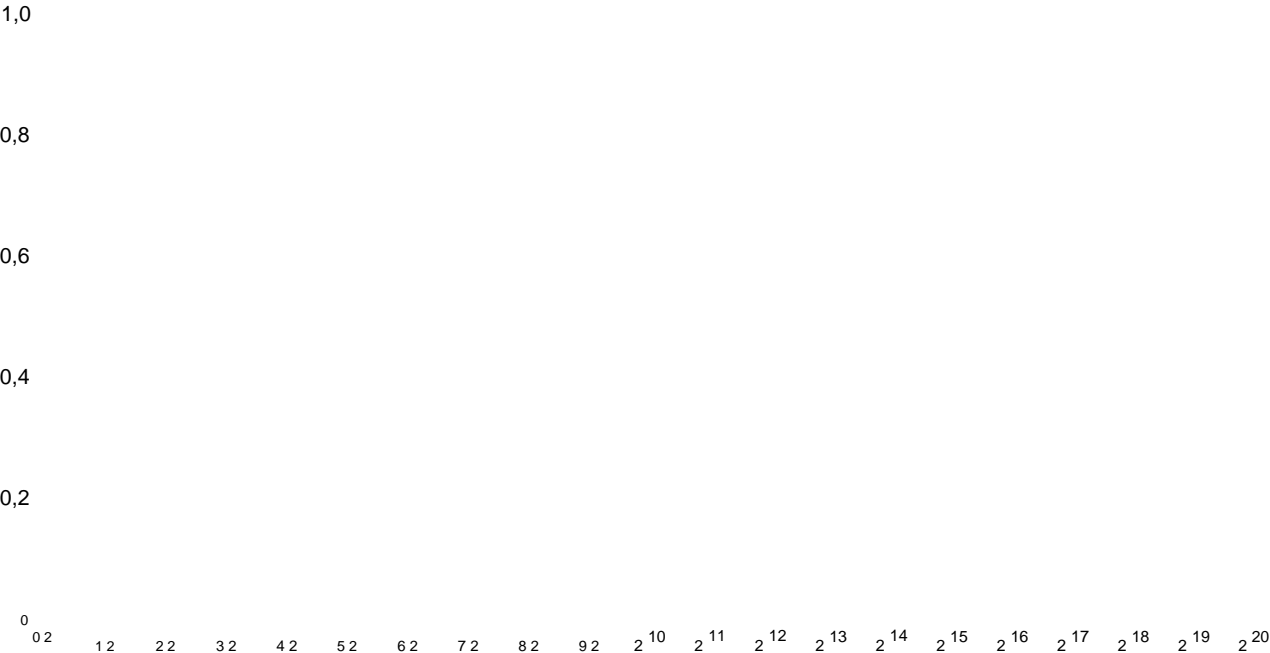


- b) Trace a variação do número médio de erros (Avg Misses) com o tamanho da passada, para cada dimensão considerada do cache de dados L1 (8kB, 16kB, 32kB e 64kB).

Observe que você poderá preencher essas tabelas e gráficos (bem como os seguintes deste relatório) em seu computador e enviar a versão impressa.

Tamanho da matriz	Stride	Avg	perde tempo	médio de ciclo
8kBytes	1			
	2			
	4			
	8			
	16			
	32			
	64			
	128			
	256			
	512			
	1024			
	2048			
16 kBytes	4096			
	1			
	2			
	4			
	8			
	16			
	32			
	64			
	128			
	256			
	512			
	1024			
	2048			
	4096			
	8192			

Tamanho da matriz	Stride	Avg	perde tempo	médio de ciclo
32kBytes	1			
	2			
	4			
	8			
	16			
	32			
	64			
	128			
	256			
	512			
	1024			
	2048			
	4096			
	8192			
	16384			
64 kBytes	1			
	2			
	4			
	8			
	16			
	32			
	64			
	128			
	256			
	512			
	1024			
	2048			
	4096			
	8192			
	16384			
	32768			



c) Analisando os resultados obtidos:

- Determine o tamanho do cache de dados L1. Justifique sua resposta.

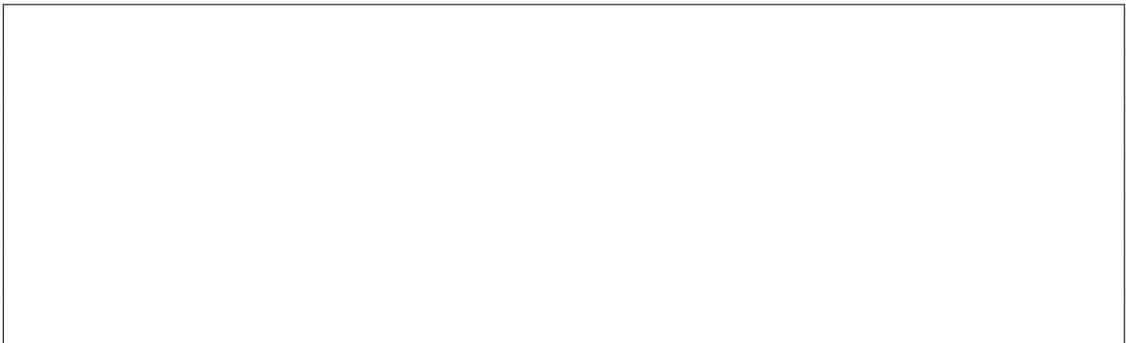
- Determine o tamanho do bloco adotado neste cache. Justifique sua resposta.

- Caracterizar o tamanho do conjunto de associatividade adotado nesta cache. Justifique sua resposta.

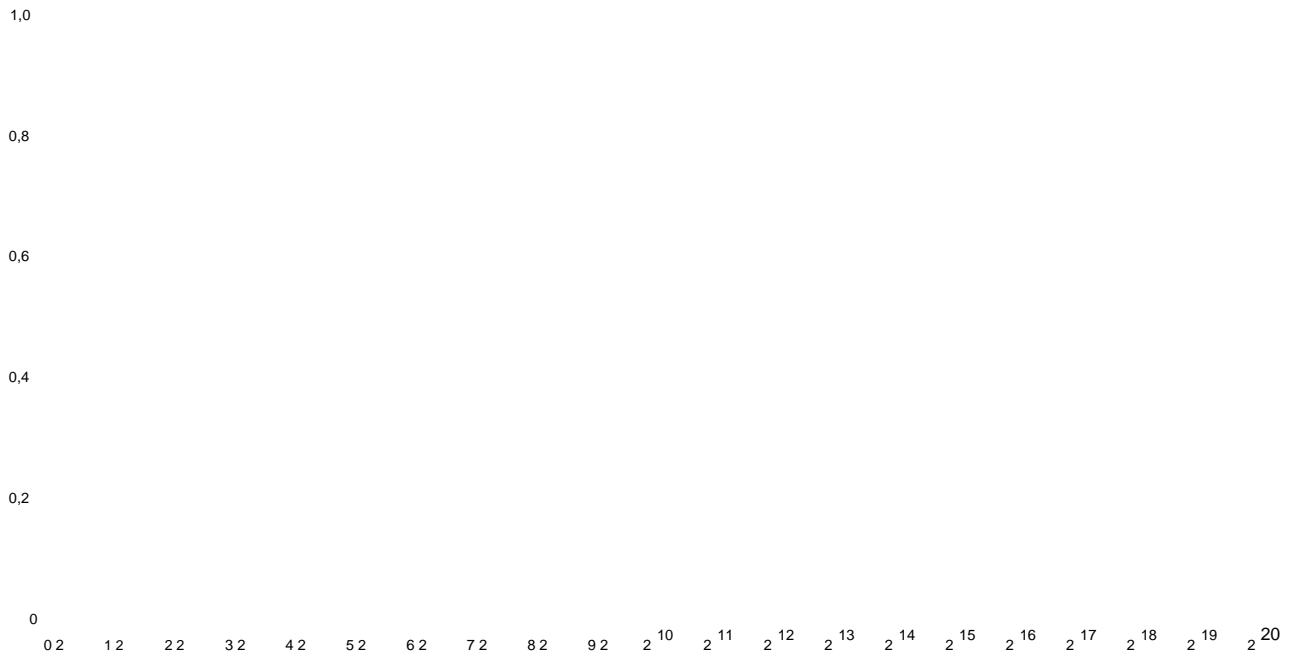


3.1.2 Modelando o Cache L2

a) Descrever e justificar as alterações introduzidas neste programa.



b) Trace a variação do número médio de erros (Avg Misses) com o tamanho da passada, para cada dimensão considerada do cache L2.



c) Analisando os resultados obtidos:

- Determine o tamanho do cache L2. Justifique sua resposta.

- Determine o tamanho do bloco adotado neste cache. Justifique sua resposta.

- Caracterizar o tamanho do conjunto de associatividade adotado nesta cache. Justifique sua resposta.

3.2 Criação de perfil e otimização de acessos ao cache de dados

3.2.1 Implementação direta

a) Qual é a quantidade total de memória necessária para acomodar cada uma dessas matrizes?

b) Preencha a tabela a seguir com os dados obtidos.

Número total de perdas de cache de dados L1	×106
Número total de instruções de carregamento/armazenamento concluídas	×106
Número total de ciclos de clock	×106
Tempo decorrido	segundos

c) Avalie a taxa de acerto do cache de dados L1 resultante:

3.2.2 Primeira Otimização: Transposição de matriz antes da multiplicação [2]

a) Preencha a tabela a seguir com os dados obtidos.

Número total de perdas de cache de dados L1	×106
Número total de instruções de carregamento/armazenamento concluídas	×106
Número total de ciclos de clock	×106
Tempo decorrido	segundos

b) Avalie a taxa de acerto do cache de dados L1 resultante:

c) Preencha a tabela a seguir com os dados obtidos.

Número total de perdas de cache de dados L1	×106
Número total de instruções de carregamento/armazenamento concluídas	×106
Número total de ciclos de clock	×106
Tempo decorrido	segundos

Comente os resultados obtidos ao incluir a transposição da matriz no tempo de execução:

d) Comparar os resultados obtidos com aqueles que foram obtidos para a implementação simples, calculando a diferença entre as taxas de acerto resultantes ($\Delta \text{HitRate}$) e os speedups obtidos.

$\Delta \text{HitRate} = \text{HitRate}_{mm2} - \text{HitRate}_{mm1}$:
$\text{Aceleração}(\# \text{Clocks}) = \# \text{Clocks}_{mm1} / \# \text{Clocks}_{mm2}$:
$\text{Aceleração}(\text{Tempo}) = \text{Tempo}_{mm1} / \text{Tempo}_{mm2}$:
Comente:

3.2.3 Segunda Otimização: Multiplicação de matriz bloqueada (ladrilhado) [2]

a) Quantos elementos da matriz podem ser acomodados em cada linha de cache?

--

b) Preencha a tabela a seguir com os dados obtidos.

Número total de perdas de cache de dados L1	×106
Número total de instruções de carregamento/armazenamento concluídas	×106
Número total de ciclos de clock	×106
Tempo decorrido	segundos

c) Avalie a taxa de acerto do cache de dados L1 resultante:

--

d) Comparar os resultados obtidos com aqueles que foram obtidos para a implementação simples, calculando a diferença entre as taxas de acerto resultantes (\bar{y} HitRate) e o speedup obtido.

\bar{y} HitRate = HitRatemm3 - HitRatemm1:
Aceleração(#Clocks) = #Clocks _{mm1} /#Clocks _{mm3} :
Comente:

e) Comparar os resultados obtidos com os obtidos para a implementação da transposição de matrizes calculando a diferença entre as taxas de acerto resultantes (\bar{y} HitRate) e o speedup obtido.

Se a aceleração obtida for positiva, mas a diferença entre as taxas de acerto resultantes for negativa, como você explica a melhoria de desempenho? (Dica: estude as taxas de acerto do cache L2 para ambas implementações;)

$\Delta \text{HitRate} = \text{HitRate}_{mm3} - \text{HitRate}_{mm2}$:
$\text{Aceleração}(\# \text{Clocks}) = \# \text{Clocks}_{mm2} / \# \text{Clocks}_{mm3}$:
Comente:

3.2.3 Comparando resultados com as especificações da CPU

Agora que você caracterizou o cache em seu computador de laboratório, compare-o com as especificações do fabricante. Para isso você pode verificar a ficha técnica do dispositivo, ou utilizar o comando `lscpu`. Comente os resultados.

--

Uma PAPI – Interface de Programação de Aplicativos de Desempenho

O projeto PAPI [1] especifica uma Interface de Programação de Aplicativo (API) padrão para acessar contadores de desempenho de hardware disponíveis na maioria dos microprocessadores modernos. Esses contadores existem como um pequeno conjunto de registros que contam Eventos, definidos como ocorrências de sinais específicos relacionados à função do processador (como falhas de cache e operações de ponto flutuante), enquanto o programa é executado no processador. O monitoramento desses eventos pode ter diversos usos na análise e ajuste de desempenho de uma aplicação, pois facilita a correlação entre a estrutura do código-fonte/objeto e a eficiência do mapeamento real de tal código para a arquitetura subjacente. Além da análise de desempenho e ajuste manual, essas informações também podem ser usadas na otimização do compilador, depuração, benchmarking, monitoramento e modelagem de desempenho.

O PAPI foi implementado em diversas plataformas diferentes, incluindo: Alpha; MIPS R10K e R12K; AMD Athlon e Opteron; Intel Pentium II, Pentium III, Pentium M, Pentium IV, Itanium 1 e Itanium 2; IBM Power 3, 4 e 5; Célula; Sun UltraSparc I, II e III, etc.

Embora cada processador tenha vários eventos nativos daquela arquitetura específica, o PAPI fornece uma abstração de software desses eventos nativos dependentes da arquitetura em uma coleção de eventos predefinidos, também conhecidos como eventos predefinidos, que definem um conjunto comum de eventos considerados relevantes. É útil para ajuste de desempenho de aplicativos. Esses eventos são normalmente encontrados em muitas CPUs que fornecem contadores de desempenho. Eles dão acesso à hierarquia de memória, eventos do protocolo de coerência de cache, contagens de ciclos e instruções, unidade funcional e status do pipeline. Consequentemente, os eventos predefinidos podem ser considerados como mapeamentos de nomes simbólicos (nome predefinido PAPI) para definições específicas da máquina (eventos contáveis nativos) para um recurso de hardware específico. Por exemplo, Total Cycles (no modo de usuário) é mapeado em PAPI_TOT_CYC. Algumas predefinições são derivadas das métricas de hardware subjacentes. Por exemplo, o Total de Perdas de Cache L1 (PAPI_L1_TCM) é a soma das Perdas de Dados L1 e das Instruções L1 em uma determinada plataforma. A lista de eventos predefinidos e nativos disponíveis em uma plataforma específica pode ser obtida executando os comandos `papi_avail` e `papi_native_avail`, ambos fornecidos pela distribuição fonte do papi.

Além do conjunto padrão de eventos para ajuste de desempenho do aplicativo, a especificação PAPI também inclui conjuntos de rotinas de alto e baixo nível para acessar os contadores. A interface de alto nível consiste em oito funções que facilitam a introdução ao PAPI, simplesmente fornecendo a capacidade de iniciar, parar e ler conjuntos de eventos. Esta interface destina-se à aquisição de medições simples, mas precisas, por engenheiros de aplicação [3, 4]:

- `PAPI_num_counters` – obtém a quantidade de contadores de hardware disponíveis no sistema; • `PAPI_flops` – chamada simplificada para obter Mflops/s (taxa de operação de ponto flutuante), real e processador tempo;
- `PAPI_ipc` – obtém instruções por ciclo, tempo real e de processador; • `PAPI_accum_counters` – adiciona contagens atuais ao array e zera contadores; • `PAPI_read_counters` – copia as contagens atuais para o array e zera os contadores; • `PAPI_start_counters` – inicia a contagem de eventos de hardware; • `PAPI_stop_counters` – interrompe contadores e retorna contagens atuais.

A seguir está um exemplo de código simples de uso da API de alto nível [3, 4]:

```
#include <papi.h>

#define NUM_FLOPS 10000 #define
NUM_EVENTS 1

int main(){ int
    Eventos[NUM_EVENTS] = {PAPI_TOT_INS}; valores
    longos_longos[NUM_EVENTS];

    /* Começa a contar eventos */ if
    (PAPI_start_counters(Events, NUM_EVENTS) != PAPI_OK) handle_error(1);

    faça algum trabalho();

    /* Ler os contadores */ if
    (PAPI_read_counters(values, NUM_EVENTS) != PAPI_OK) handle_error(1);

    printf("Depois de ler os contadores: %lld\n", values[0]);

    faça algum trabalho();

    /* Adicione os contadores */ if
    (PAPI_accum_counters(values, NUM_EVENTS) != PAPI_OK) handle_error(1);

    printf("Depois de adicionar os contadores: %lld\n", valores[0]);

    faça algum trabalho();

    /* Para de contar eventos */ if
    (PAPI_stop_counters(values, NUM_EVENTS) != PAPI_OK) handle_error(1);

    printf("Depois de parar os contadores: %lld\n", valores[0]);
}
```

Saída possível:

```
Depois de ler os contadores: 441027
Depois de adicionar os contadores: 891959
Depois de parar os contadores: 443994
```

A interface de baixo nível totalmente programável oferece opções mais sofisticadas para controlar os contadores, como definir limites para interrupção em caso de overflow, bem como acesso a todos os modos e eventos de contagem nativos. Essa interface é destinada a criadores de ferramentas terceirizados ou usuários com necessidades mais sofisticadas.

A especificação PAPI também fornece acesso aos temporizadores mais precisos disponíveis na plataforma em uso. Esses temporizadores podem ser usados para obter tempo real e virtual em cada plataforma suportada: o relógio de tempo real funciona o tempo todo (por exemplo, um relógio de parede), enquanto o relógio de tempo virtual funciona apenas quando o processador está funcionando no modo de usuário.

No exemplo de código a seguir, `PAPI_get_real_cyc()` e `PAPI_get_real_usec()` são usados para obter o tempo real necessário para criar um conjunto de eventos em ciclos de clock e em microssegundos, respectivamente [3, 4]:

```
#include <papi.h>

int main(){ longo
    longo start_cycles, end_cycles, start_usec, end_usec; int EventSet = PAPI_NULL;

    se (PAPI_library_init(PAPI_VER_CURRENT) != PAPI_VER_CURRENT)
        saída(1);

    /* Cria um EventSet */ if
    (PAPI_create_eventset(&EventSet) != PAPI_OK)
        saída(1);

    /* Obtém a hora de início em ciclos de clock */ start_cycles =
    PAPI_get_real_cyc();

    /* Obtém a hora de início em microssegundos */ start_usec =
    PAPI_get_real_usec();

    faça algum trabalho();

    /* Obtém a hora final em ciclos de clock */ end_cycles =
    PAPI_get_real_cyc();

    /* Obtém o horário final em microssegundos */ end_usec =
    PAPI_get_real_usec();

    printf("Ciclos de relógio de parede: %lld\n", end_cycles - start_cycles); printf("Tempo do relógio de parede
    em microssegundos: %lld\n", end_usec - start_usec);
}
```

Saída possível:

Ciclos do relógio de parede: 100173
Tempo do relógio de parede em microssegundos: 136