ISEL | DEETC

Programação de Sistemas Computacionais

Serie 03 de Exercícios

Semestre de Verão de 2009/2010

Autor:
31401 – Nuno Cancelo

Indicie

Prefácio

Ao ter sido dado a conhecer saída deste trabalho, fiquei preocupado em não me ser possível implementa-lo e apresenta-lo dentro do tempo estipulado. Apesar da implementação não ser demorada ou mesmo complicada, era necessário mais tempo do que teria com o prazo que teoricamente estaria estipulado.

Com a publicação do trabalho, verifiquei que o prazo tinha sido alargado, não quero com isto insinuar que o prazo foi alargado devido às razões que apresentei, e que muito provavelmente até foi coincidência, no entanto que agradecer pelo prazo apresentado.

Muito obrigado.

Exercício 1

1. Dado um sistema de cache de 16KiB, 4-way set associative, com linhas de 32 bytes, indique (apresentando os respectivos cálculos, quando se aplique):

a)
Caracterize a divisão dos endereços em tag, index e offset, sabendo que a cache é utilizada num sistema com um address bus de 36 bits.

E = 4-way
B = 32 bytes
m = 36 bits
C = 16 KiB

$$C = S * E * B \Leftrightarrow 16 * 1024 = S * 4 * 32 \Leftrightarrow \frac{16 * 1024}{4 * 32} = S \Leftrightarrow S = 128$$
$$s = \log_2(S) \Leftrightarrow s = 7$$
$$b = \log_2(B) \Leftrightarrow b = 5$$
$$t = m - (s + b) \Leftrightarrow t = 36 - (7 + 5) \Leftrightarrow t = 24$$
$$M = 2^m \Leftrightarrow M = 4294967296\ endereços$$

t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	t	s	s	s	s	s	s	s	b	b	b	b	b

Resposta:
Para os elementos fornecidos, os tamanhos obtidos são:
TAG: 24 bits
INDEX: 07 bits
OFSSET: 05 bits
---------- ----
Total: 36 bits

b)
Considere sizeof(int) == 4, a variável global int dat[N] e a função cyclic_read, que consiste num ciclo infinito de leitura de todas as posições do array dat e que utiliza apenas os registos do processador para manter as variáveis auxiliares de que necessita.

1. Qual o menor valor de N que garante que duas entradas do array dat pertencem ao mesmo set da cache?

Dados fornecidos:
* sizeof(int) == 4 bytes
* int dat[N]
* função cyclic_read.

Dados pretendidos:

- N mínimo, que garante a partir do qual que duas entradas de dat, pertencem ao mesmo set (em vias diferentes)?
- N mínimo, que garante a ocorrência de cache misses na segunda passagem no array em diante?

Raciocínio:
Admitindo (para efeitos de simplificação), que:
* todos os SET's e vias estão vazios
* o endereço base do array se encontra num endereço múltiplo de 8

E tenho que:
* cada linha de cache tem 32 bytes = 8 * 4 bytes = 8 inteiros
* cada via da cache tem 128 sets

Dado que é lido o tamanho da linha (quando possível) na obtenção de dados, que não estão presentes na cache, verifica-se que após a obtenção de 32 bytes, a próxima linha a "carregar" na cache será colocada no SET seguinte e assim sucessivamente até preencher todos os SET's dessa via.
Assim sendo, temos que S*B=espaço ocupado por uma via. Após o preenchimento total desta via, o próximo a ser "carregado" será colocado no primeiro SET, numa das outras vias ainda disponíveis.

Assumindo este pressuposto, verifico que:
S * B = 128 * 32 = 4098 bytes = 1024 inteiros
Desta forma o 1025º inteiro do array, é garantido, que irá ser colocado no mesmo SET dos primeiros 8 inteiros, mas noutra via.

Resposta:
Desta forma, o N mínimo que vai de encontro ao solicitado no enunciado será 1025.

---

2. E qual o menor valor de N que garante a ocorrência de cache misses da segunda passagem no array em diante, mesmo que existam caches separadas para código e dados?

Ao continuar o estudo da cache, e de forma análoga, demonstra-se que para obter uma cache completamente cheia temos:

S * B * E = 128 * 32 * 4 = 16384 bytes = 4096 inteiros

Desta forma corrobora-se o facto de a cache ficar totalmente preenchida, a partir deste momento todos os valores "carregados" vai originar misses e proceder à substituição de dados nas linhas de SET da cache.

Resposta:
Concluí-se então que o N mínimo que garante a ocorrência de caches misses será com N= 4097.

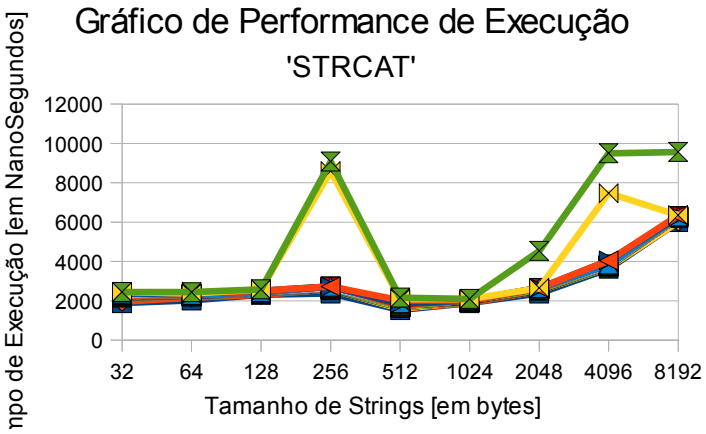
Exercício 2

Considere as implementações das funções xstrcat e astrcat, realizadas nas séries de exercícios anteriores, bem como as implementações x1strcat e x2strcat apresentadas a seguir e a implementação de strcat da biblioteca standard, disponibilizada pelas ferramentas de desenvolvimento que utiliza. Meça o tempo de execução destas cinco funções para strings s1 e s2 com dimensões iguais, compreendidas entre 16B e 8KiB em passos de 16 bytes, apresentando os resultados em tempo absoluto de forma gráfica. Comente os aspectos relevantes dos resultados.

Os testes de performance destas funções foi realizados num computador com as seguintes características:

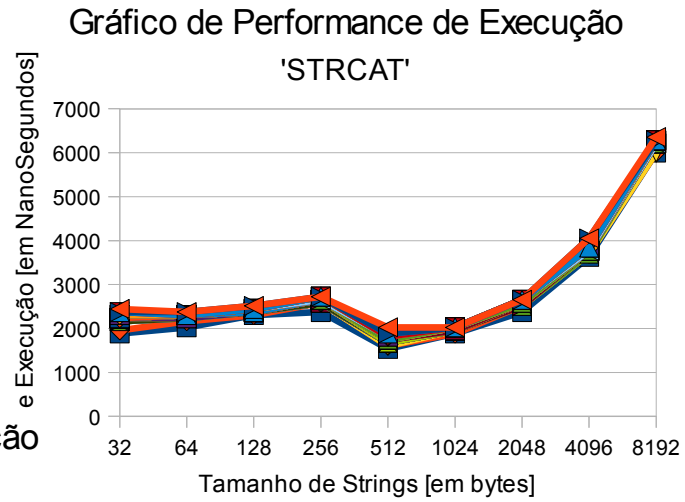
- S.O. : Ubuntu (9.10 Karmic) (dedicado)
- Memória: 2.0 GiB
- Processador: Intel Core 2 Duo (T7500@2.20GHz)
- Com alguns programas a correr em simultâneo (Messenger, MediaPlayer, Web Browser)

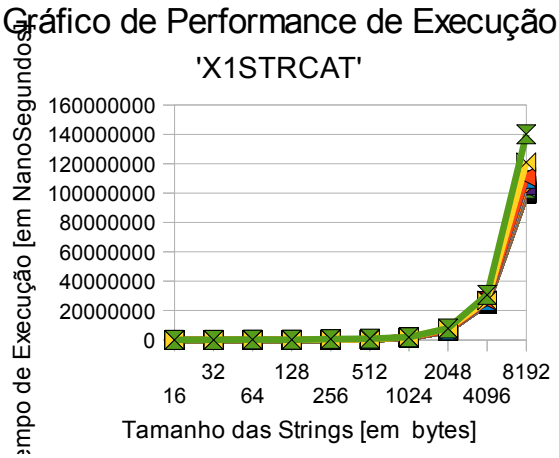
Para a execução destes testes foram utilizadas strings iguais com tamanho inicial igual a 16 bytes e o seu tamanho a duplicar para os testes seguintes até ao tamanho máximo de 8192 bytes. Desta forma foi possível testar 10 strings de tamanhos diferentes. Para cada um destes testes foram executados 64 ciclos, e os resultados guardados e ordenados de forma crescente.
Os resultados obtidos foram exportados para uma folha de cálculo, de onde foram gerados os gráficos apresentados.



Performance do "STRCAT"
Verifica-se que houve alturas que demorou bastante mais tempo a realizar a sua tarefa, esta situação deve-se ao facto dos processos em concorrência geridos pelo sistema operativo.

Verifica-se que ao retirarmos os testes que fogem do padrão, o resultado demonstra-se mais estável, tendo um crescimento deveras exponencial com strings com mais de 512 bytes.

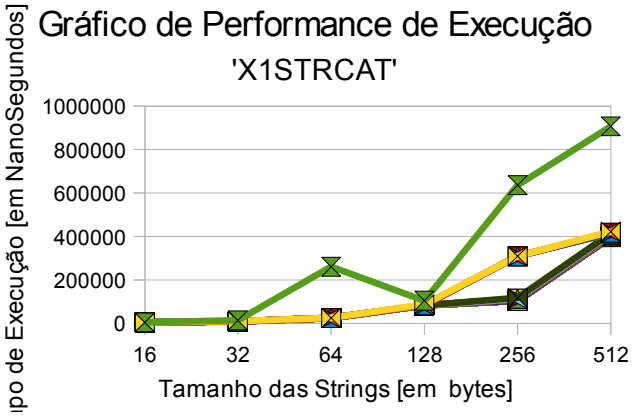




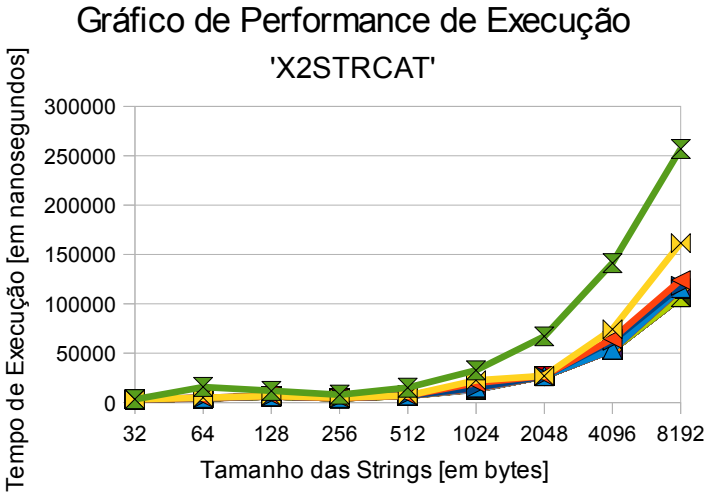
Performance "X1STRCAT"

Aqui verifica-se que a performance foi bastante degradada, principalmente a partir de strings com tamanhos acima de 1024 bytes.
Este facto justifica-se pelo facto de se estar a continuamente a chamar uma função que devolve o tamanho da string, e que para esse efeito é necessário percorrer toda a string para obter esse valor.

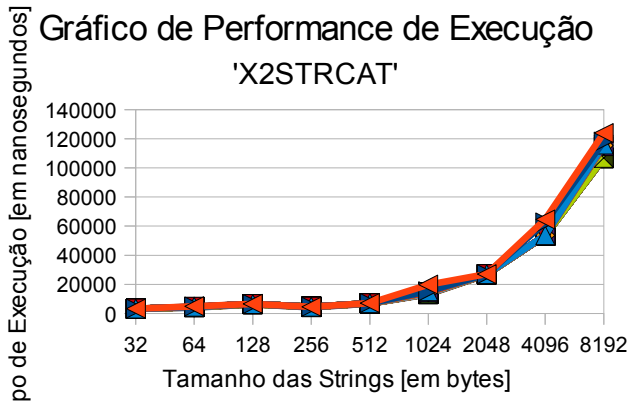
Verfica-se que para valores inferiores as 1024 bytes, o seu crescimento é quase linear, mas ainda assim bastantes superior ao que foi conseguido pelo "STRCAT", devido ao uso indevido de funções auxiliares.



Performance "X2STRCAT"



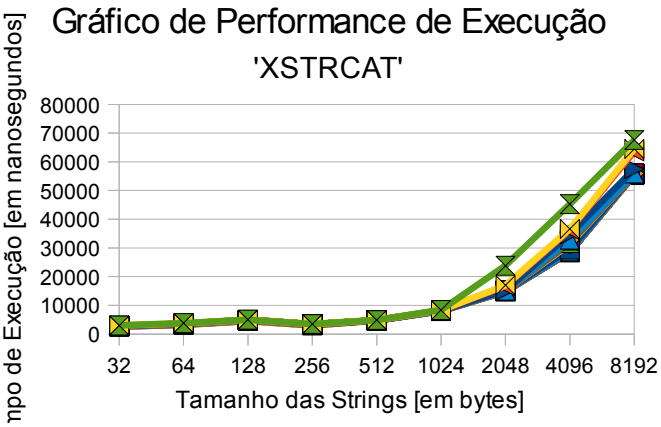
Perante este gráfico nota-se uma melhoria significativa face aos anteriores. O tempo de execução foi reduzido, devido à utilização da recursividade como forma de concatenação de strings, originando ao mesmo tempo o uso da inúmeras stack frames, gastando mais memoria para o efeito.

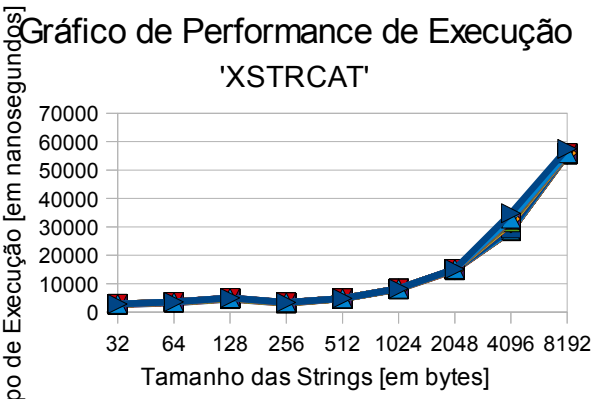


Ao retirarmos os testes que fogem ao padrão nota-se uma grande estabilidade nos resultados, obtendo-se tempos abaixo dos 140000 nano-segundos para as maiores strings.

Performance "XSTRCAT"

Os testes referentes a implementação apresentada nas series anteriores, apresenta uma grande melhoria na execução, uma vez que a sua implementação se tentou ser o mais eficiente possível, não gastando mais memória do que a necessária.
Ainda assim, a implementação da biblioteca prova-se mais rápida.

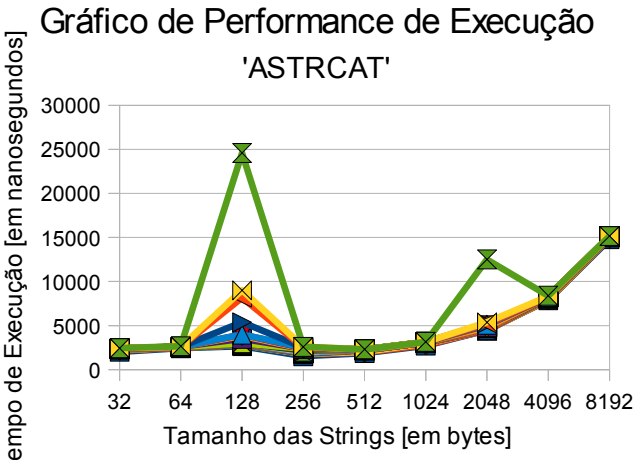


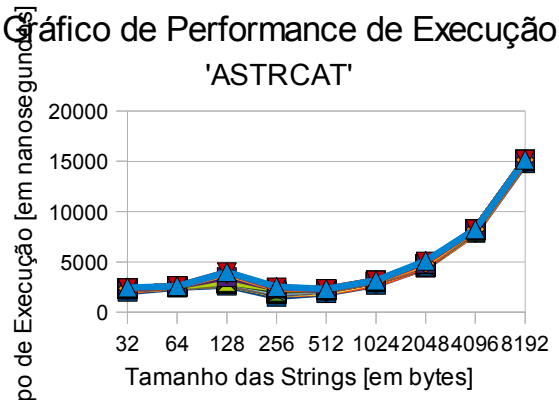


A implementação apresentada, em termos gerais apresenta-se abaixo dos 60000 nano-segundos nas strings maiores.

Performance “ASTRCAT”

A versão da implementação em assembly apresenta uma enorme melhoria nos resultados, uma vez que se detém um maior controlo sobre o fluxo de informação nos registos do processador.





A remoção dos piores tempos, demonstra que os tempos ficam abaixo dos 16000 nano-segundos, sendo a performance geral bastante estável.

Conclusão

Durante a implementação desta série de exercícios consolidou-se o conhecimento sobre a forma como as caches (nomeadamente “set associative”) funcionam e a sua importância para a implementação de algoritmos.

Embora não tenhamos muito controlo sobre a localização dos nossos dados na Cache, é-nos possível originar código de forma que os mesmos sejam dispostos nas mesmas vias da cache, evitando que sejam originados caches-misses, respeitando os princípios da localidade temporal e espacial, podemos de certa forma aumentar a probabilidade de os nossos dados serem carregados para a cache, sem ter a necessidade de voltar a carrega-los.

Os testes de performance demonstraram como várias implementações da mesma solução podem ser mais ou menos eficazes, atendendo que o CPU não está dedicado ao nosso programa e que vários programas do Sistema Operativo também tem processos a decorrer e que vão solicitar a suspensão temporária do processamento.

A implementação de soluções, e o seu sucesso, dependem da forma como as mesma são implementadas, e o conhecimento da arquitectura destino e do Sistema Operativo, permitem que seja implementada uma solução mais eficiente de acordo com o sistema destino.

Bibliografia

- Computer Systems, A Programmer's Perspective 2nd Edition
  - Randal E. Bryant, David R. O'Hallaron
    Editora: Prentice Hall
    - Cap. 6: The Memory Hierarchy