UTILIZAÇÃO DAS EXTENSÕES MULTIMÍDIA DOS PROCESSADORES INTEL® PARA REDUÇÃO DO NÚMERO DE CICLOS PARA A EXECUÇÃO DE PROGRAMAS

HOLANDA, Adriano de Jesus[1]
RUIZ, Evandro Eduardo Seron[2]
CARNEIRO, Antonio Adilton Oliveira[2]

Recebido em: 2013.03.11 **Aprovado em:** 2014.02.14 **ISSUE DOI:** 10.3738/1982.2278.878

RESUMO: A utilização das extensões multimídias com registradores que realizam a mesma operação sobre vários dados ao mesmo tempo dos atuais processadores podem reduzir o tempo de execução de programas que lidam com operações aritméticas sobre grande quantidade de dados. O objetivo deste trabalho foi comparar o número de ciclos durante a execução de dois programas utilizados para o cálculo da correlação cruzada em duas dimensões para séries de diferentes tamanhos. Um programa foi implementado utilizando a extensão SSE dos processadores Intel® de 64 bits e o outro foi implementado sem o uso da extensão. A comparação entre os programas demonstrou que o programa usando a extensão SSE utilizou 38,37% menos ciclos de processador que a mesma implementação escrita sem uso da extensão SSE.

Palavras-chave: Assembly. Processador. SIMD. Correlação cruzada.

USE OF MULTIMEDIA EXTENSIONS OF INTEL® PROCESSORS TO DECREASE THE NUMBER OD CYCLES OF A PROGRAM EXECUTION

SUMMARY: The use of multimedia extension with registers that perform the same operation in multiple data of the current processors should decrease the execution time of programs used to perform the same operation in a large quantity of data. The aim of the work was quantify the number of cycles needed to perform two-dimensional cross correlation calculation on a number of generated series with different number of elements, using a program compiled using Intel® Assembly x86-64 language and SSE extension and another one compiled without SSE extension. The program using SSE performed the calculation using in average 38.37% less processor cycles than the program without SSE.

Keywords: Assembly. Processor. SIMD. Cross correlation.

INTRODUÇÃO

Mesmo com a disseminação dos processadores com arquitetura de 64 bits substituindo os de 32 bits, as demandas por maior capacidade de armazenamento temporário nos registradores e melhor eficiência na operação sobre os dados armazenados vêm pressionando os fabricantes de processadores a adotarem soluções que se desviam da arquitetura normalmente empregada para construção dos processadores.

A principal modificação é a inclusão de registradores de 128 e 256 bits para a realização de vários cálculos de uma única vez, aumentando a performance dos programas, que utilizam estes recursos, causado pela redução no número de ciclos durante o processamento.

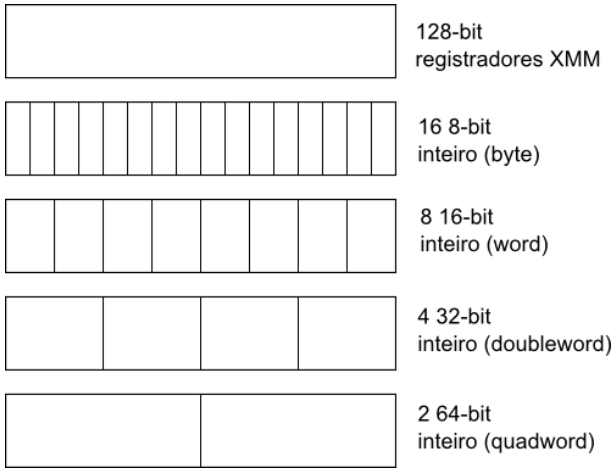
Estes recursos que começaram a ser manufaturados pela Intel® permitem que múltiplos dados

[1]FAFRAM. Departamento de Computação e Matemática na Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto (FFCLRP) - Universidade de São Paulo (USP)
[2] Departamento de Computação e Matemática - FFCLRP/USP

sejam manipulados por uma única instrução (SIMD – *Single Instruction Multiple Data*), e fazem parte da tecnologia de extensão multimídia (MMX – *Multimedia Extension*) introduzida a partir do Pentium II, para realizar operações complexas sobre números inteiros.

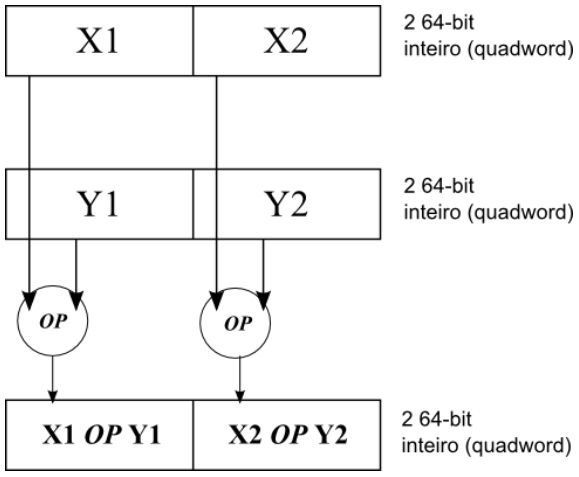
Uma extensão da tecnologia SIMD denominada SSE (*Streaming* SIMD *Extension*) possui registradores de 128 bits para o armazenamento de múltiplos dados. A Figura 1 mostra como dados do tipo inteiro de diferentes tamanhos podem ser armazenados em um registrador SSE (BLUM, 2005).

Figura 1 - Possibilidade de divisões dos registradores XMM de 128-bit para armazenamento de números inteiros. Fonte: Blum (2005).



Utilizando os registradores de 128-bit para armazenamento de 2 números inteiros de 64 bits, que poderiam representar 2 pixels de um mapa de bits de imagens no padrão RGB (*Red, Green, Blue*), a soma de 2 números dois a dois pode ser realizada em 1 ciclo do processador, enquanto utilizando o método tradicional seriam necessárias dois ciclos do processador para a execução da operação. A Figura 2 ilustra como a operação em dados múltiplos é utilizada para redução do número total de ciclos para o processamento.

Figura 2 - Operação realizada em múltiplos dados sobre o mesmo registrador.



A Figura 2 ilustra a execução de uma mesma operação em vários dados, onde a operação *OP*

é realizada sobre os primeiros segmentos de cada registrador e o resultado armazenado no primeiro segmento do registrador destino, enquanto a mesma operação é realizada sobre os segundos segmentos com o armazenamento ocorrendo no segundo segmento do registrador destino. Este é o modo de operação das instruções Assembly `paddq` e `pmuldq`, adição e multiplicação de *quadwords* empacotados, respectivamente, utilizadas em nosso trabalho (INTEL®, 2012).

A utilização destes recursos para a execução de operações aritméticas pode reduzir o número de ciclos necessários para completar um tarefa, reduzindo o tempo de execução do programa.

O objetivo deste trabalho foi comparar o número de ciclos de processador durante a execução das instruções de dois programas, um que utiliza a extensão SSE com registradores de 128 bits, e outro sem o uso das extensão SSE.

MATERIAL E MÉTODOS

O algoritmo para o cálculo de correlação cruzada entre duas séries foi escolhido para a realização da comparação. A correlação cruzada é comumente utilizada em processamento de sinais e imagens para analisar as similaridades entre as séries de dados que podem ser valores de pixels ou intensidade do sinal em determinado tempo. A correlação cruzada pode ser calculada usando a equação

(1)

$$r_{xy} = \frac{\sum (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum (x_i - \overline{x})^2 \sum (y_i - \overline{y})^2}}$$

onde a média dos valores é calculada por

$$\overline{x} = \frac{\sum x_i}{n} \qquad (2)$$

Porém, a Equação 1 tem complexidade $O(n^2)$, onde n é o número de elementos na série, pois há a necessidade de realizar duas passagens pelos dados, uma para calcular a média das séries x e y, e outra para calcular o valor da correlação cruzada r_{xy} entre as séries.

A Equação 1 pode ser decomposta de modo a necessitar somente de uma passagem pelos dados, possuindo complexidade $O(n)$, e resultando na Equação 3:

(3)

$$r_{xy} = \frac{n \sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n \sum x_i^2 - (\sum x_i)^2} \sqrt{n \sum y_i^2 - (\sum y_i)^2}}$$

A implementação da Equação 3 sem o uso da extensão SSE é mostrada na

Figura ***3***. Esta foi feita utilizando a linguagem C e segue a sugerida por Seyfard (2012), com modificações feitas pelos autores para operar sobre números inteiros:

Figura 3- Código em linguagem C do cálculo da correlação cruzada entre duas séries de dados.

```
1   double xcorr(long x[], long y[], n) {
2           register long i;
3           register long sum_x=0, sum_y=0,
4                                 sum_xx=0, sum_yy=0, sum_xy=0;

               5           for (i = 0; i < n ; i++) {
6                register long xval = *(x+i);
7                register long yval = *(y+i);
8                sum_x += xval;
9                sum_y += yval;
10               sum_xx += xval * xval;
11               sum_yy += yval * yval;
12               sum_xy += xval * yval;
13          }
14          return ((double)(n*sum_xy-sum_x*sum_y))/
15                    (sqrt((n*sum_xx-sum_x*sum_x)*
16                    (n*sum_yy-sum_y*sum_y)));
17  }
```

O programa que utiliza a extensão SSE é mostrado na Figura 5 (Apêndice). Este foi implementado em Assembly para processadores Intel® x86 de 64 bits. O código também segue, em linhas gerais, o descrito por Seyfarth (2012), com duas modificações. A primeira é utilizar somente números inteiros para as séries, pois normalmente os valores dos pixels na imagem são do tipo inteiro. A segunda modificação é uma verificação no cálculo final da correlação para evitar divisões por zero, que poderiam causar erro fatal ao final da execução do programa. Os códigos podem ser obtidos no site https://github.com/ajholanda/nixus.

O programa em C foi compilado usando o gcc (GNU *Compiler Collection*, http://gcc.gnu.org/) sem nenhum parâmetro adicional passado ao compilador. O código em Assembly (Figura 5) foi compilado usando o montador Yasm (http://yasm.tortall.net/) com os seguintes parâmetros: "-Worphan-labels -f elf64 -g dwarf2 -l corr.lst". Os programas foram executados em um computador com processador Intel® Core™ 2 Duo com 2,93 GHz e 3MB de memória cache, e memória DRAM com 3GB de capacidade.

O número de valores atribuídos para as séries foram gerados a partir dos índices do arranjo contendo os elementos com as seguintes atribuições:

```
x[i] = 1+i;
y[i] = 1+2*i;
```

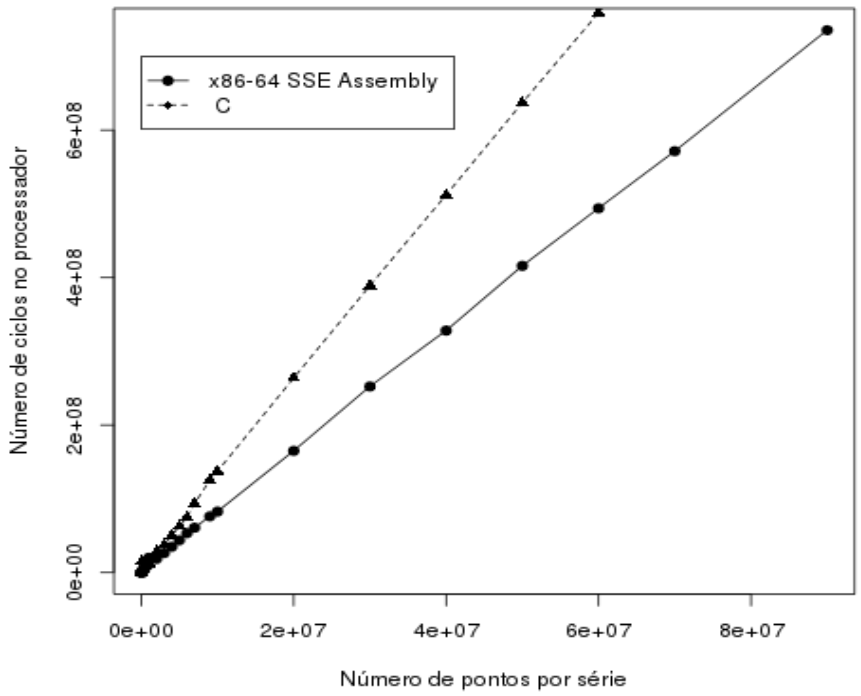
onde `i` é o índice do arranjo onde está sendo efetuado o *loop*, `x` é o arranjo que contém os pontos da primeira série e `y` é o arranjo que contém os elementos da segunda série.

Os cálculos de correlação entre as séries foram efetuados para os seguintes números de elementos: {<1.000, 1.000, 9.000>, <10.000, 10.000, 90.000>, <1.000.000, 1.000.000, 9.000.000>, <10.000.000, 10.000.000, 90.000.000>}. O primeiro valor da tupla indica valor inicial para o *loop*, o segundo o passo e o terceiro o limite do *loop*.

RESULTADOS

A Figura 4 contêm a relação entre os número de pontos das séries x e y e o número de ciclos utilizados para calcular a correlação entre as séries utilizando o programa em Assembly com extensão SSE e em C sem extensão SSE.

Figura 4- Número de ciclos utilizados pelo processador em função de número de elementos das séries geradas para o cálculo da correlação cruzada.



Em média houve uma redução de 38,37% do número de ciclos utilizando o código em Assembly (SSE) quando comparado com o código em C. Para um número menor de valores na série, a performance do código em Assembly (SSE) é maior devido ao fato da menor utilização da memória cache. Para o número de elementos na série entre 10.000 e 100.000, a redução do número de ciclos usando o código em Assembly varia de 64,5% a 94,4. A partir de 100.000 o a redução dos valores de uso de ciclos do processador cai para menos de 40%.

DISCUSSÃO

Quando se tem por objetivo melhorar a performance dos programas que processam grande quantidade de dados, caso comum em processamento de imagens e sinais, o conhecimento do hardware utilizado para execução das tarefas pode contribuir para identificar pontos de otimização.

As linguagens de alto nível como o C fornecem um nível de abstração e portabilidade que contribuem para facilitar o gerenciamento do código fonte. Porém, alguns recursos recentes do processadores modernos ainda não estão disponíveis nas linguagens de programação mais comuns. Por este motivo, a integração de código de alto nível com Assembly, usando esta última em pontos que exijam maior performance pode ser uma solução para reduzir o tempo de processamento total.

O processador Intel® Core™ i7 possui registradores de 256 bits para a realização das

operações em múltiplos dados, através de sua extensão AVX *(Advanced Vector Extensions)*, possuindo operações específicas para dados multimídia.

Outros pontos de otimização mais ligados ao sistema operacional, como por exemplo melhor controle do sistema de entrada/saída (E/S), também cooperam para a redução do tempo total de processamento. Por exemplo, a manipulação do *buffer*, uso de operações de E/S não bloqueante e mapeamento do arquivo contendo os dados para memória principal podem reduzir o tempo de acesso aos dados, aumentando o fluxo de instruções no processador.

CONCLUSÃO

O uso da extensão SSE do processador Intel® para o cálculo da correlação cruzada entre duas séries de valores inteiros de diferentes tamanhos reduziu o número de ciclos utilizados no processador para a execução do programa em média 38,37%, quando comparado com o programa que não utiliza a extensão SSE.

REFERÊNCIAS

BLUM, Richard. **Professional Assembly Language.** Wrox, 2005.

INTEL® Corporation. **Intel® 64 and IA-32 Architectures Software Developer's Manual.** Volume 2B, 2012.

SEYFARTH, Ray. **Introduction to 64 Bit Intel Assembly:** Language Programming for Linux, 2012 (Kindle Edition).

APÊNDICE

Na Figura 5 o código em Assembly Intel® x86-64 é mostrado com o número da linha de cada instrução na primeira coluna, os rótulos na segunda coluna, as operações na terceira, os operandos na quarta e os comentários na quinta após o símbolo ";".

Das linhas 15 à 29 estão as instruções para a realização do somatório para o cálculo da correlação cruzada conforme mostrado na Equação 3. As instruções paddq e pmuldq executam a soma e multiplicação, respectivamente, de quatro valores inteiros conforme mostrado na Figura 4. Das linhas 30 à 62, há instruções para a preparação dos resultados das somas, convertendo-os para ponto flutuante, para efetuar a divisão, e realizando cálculo final da correlação. A linha 56 garante que se o numerador e denominador da equação forem iguais a zero, ou seja a correlação seja igual a 1, a divisão pelo denominador igual a zero, não seja realizada retornando o valor 1 sem efetuar a divisão final.

Figura 5- Código em Assembly Intel® x86-64 para o cálculo da correlação cruzada entre duas séries de números inteiros. **(Continua)**

```
1                segment   .data
2   one          dq         1
3                segment   .text
4                global    xcorr_nat
5   xcorr_nat:
6                xor       r8,r8

7                move      rcx, rdx

8                subpd     xmm0, xmm0      ; zera sum_x
9                movdqa    xmm1, xmm0      ; zera sum_y
10               movdqa    xmm2, xmm0      ; zera sum_xx
11               movdqa    xmm3, xmm0      ; zera sum_yy
12               movdqa    xmm4, xmm0      ; zera sum_xy
13               movdqa    xmm8, xmm0      ; zera n
14               movdqa    xmm9, xmm0      ; zera variável temporária
15  .loop:                                 ;; Main loop
16               movdqa    xmm5,           ; atribui valor do próximo x da série
                           [rdi+r8]
17               movdqa    xmm6,           ; atribui valor do próximo y da série
                           [rsi+r8]
18               paddq     xmm0, xmm5      ; sum_x
19               paddq     xmm1, xmm6      ; sum_y
20               movdqa    xmm7, xmm5      ; atribui x a um registrador temporário
21               pmuldq    xmm7, xmm6      ; x*y
22               pmuldq    xmm5, xmm5      ; x*x
23               pmuldq    xmm6, xmm6      ; y*y
24               paddq     xmm2, xmm5      ; sum_xx
25               paddq     xmm3, xmm6      ; sum_yy
26               paddq     xmm4, xmm7      ; sum_xy
27               add       r8, 16
28               sub       rcx, 2
```

Figura 6- Código em Assembly Intel® x86-64 para o cálculo da correlação cruzada entre duas séries de números inteiros. **(Conclusão)**

```
29                  jnz         .loop
30                  haddpd      xmm0, xmm0      ; compacta sum_x
31                  cvtdq2pd    xmm0, xmm0      ; converte sum_x para ponto flutuante
32                  haddpd      xmm1, xmm1      ; compacta sum_y
33                  cvtdq2pd    xmm1, xmm1      ; converte sum_y para ponto flutuante
34                  haddpd      xmm2, xmm2      ; compacta sum_xx
35                  cvtdq2pd    xmm2, xmm2      ; converte sum_xx para ponto flutuante
36                  haddpd      xmm3, xmm3      ; compacta sum_yy
37                  cvtdq2pd    xmm3, xmm3      ; converte sum_yy para ponto flutuante
38                  haddpd      xmm4, xmm4      ; compacta sum_xy
39                  cvtdq2pd    xmm4, xmm4      ; converte sum_xy para ponto flutuante
40                  cvtsi2sd    xmm8, rdx       ; armazena o valor de n em xmm8
41                  mulsd       xmm4, xmm8      ; n*sum_xy
42                  movsd       xmm9, xmm0      ; tmp := sum_x
43                  mulsd       xmm9, xmm1      ; sum_x*sum_y
44                  subsd       xmm4, xmm9      ; numerador := n*sum_xy - sum_x*sum_y
45                  mulsd       xmm2, xmm8      ; n*sum_xx
46                  mulsd       xmm0, xmm0      ; sum_x*sum_x
47                  subsd       xmm2, xmm0      ; denominador0 := n*sum_xx - sum_x*sum_x
48                  sqrtsd      xmm2, xmm2      ; denominador0 := sqrt(denominador0)
49                  mulsd       xmm3, xmm8      ; n*sum_yy
50                  mulsd       xmm1, xmm1      ; sum_y*sum_y
51                  subsd       xmm3, xmm1      ; denominador1 := n*sum_yy - sum_y*sum_y
52                  sqrtsd      xmm3, xmm3      ; denominador1 := sqrt(denominador1)
53                  mulsd       xmm2, xmm3      ; denominador := denominador0 * denominador1
54                  movsd       xmm9, xmm2      ; tmp := denominador
55                  comisd      xmm4,xmm9       ; denominador  == numerador ?
56                  jz          DEN_EQ_NUM      ; (denominador  == numerador) := verdadeiro
57                  divsd       xmm4, xmm2      ; correlação := numerador/denominador
58                  movsd       xmm0, xmm4      ; retorna correlação
59                  jz          EXIT
60  DEN_EQ_NUM:     movsd       xmm0, [one]     ; correlação := 1
61                  cvtdq2pd    xmm0, xmm0      ; converte inteiro para ponto flutuante
62  EXIT:
```