

Programação Paralela: das *threads* aos FPGAs

Processamento Vetorial (SIMD)

Prof. Ricardo Menotti
menotti@ufscar.br

Prof. Maurício Acconcia Dias
macccdias@gmail.com

Prof. Helio Crestana Guardia
helio.guardia@ufscar.br

Departamento de Computação
Universidade Federal de São Carlos

Atualizado em: 7 de maio de 2020

Roteiro

Histórico

Processamento Vetorial

Como usar?

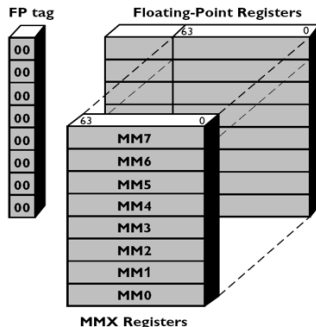
Bibliografia

Histórico

- ▶ Na década de 90 as aplicações de multimedia (imagem, som, video) se popularizaram
 - ▶ Uso de tipos de dados pequenos
 - ▶ pixels: 8-bits
 - ▶ áudio: 16-bits
 - ▶ Repetição de operações
 - ▶ Paralelismo inerente
- ▶ Surgiram recursos de processamento SIMD em várias arquiteturas.
 - ▶ MMX – Intel
 - ▶ NEON – ARM
 - ▶ AltiVec – Apple/IBM/Motorola

Histórico: MMX

- ▶ Novo tipo de dados: 64-bit packed (empacotado).
- ▶ Operações com números inteiros apenas
- ▶ Reutiliza registradores de ponto flutuante
 - ▶ Não é possível misturar instruções MMX com operações de ponto flutuante (sem grande perda de desempenho).



Processamento Vetorial

Registradores

SSE Data Types (16 XMM Registers)

__m128	Float	Float	Float	Float	4x 32-bit float										
__m128d	Double		Double		2x 64-bit double										
__m128i	B	B	B	B	B	B	B	B	B	B	B	B	B	B	16x 8-bit byte
__m128i	short	short	short	short	short	short	short	short	short	8x 16-bit short					
__m128i	int	int	int	int	4x 32bit integer										
__m128i	long long		long long		2x 64bit long										
__m128i	doublequadword														1x 128-bit quad

AVX Data Types (16 YMM Registers)

__mm256	Float	Float	Float	Float	Float	Float	Float	8x 32-bit float
__mm256d	Double		Double		Double		4x 64-bit double	
__mm256i	256-bit Integer registers. It behaves similarly to __m128i. Out of scope in AVX, useful on AVX2							

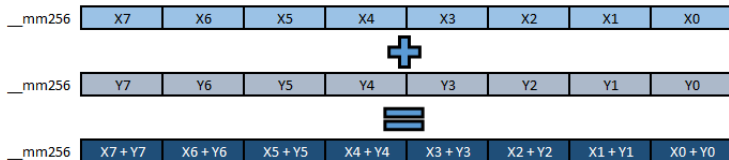
Processamento Vetorial

Funcionamento

Executam a mesma operação em múltiplos elementos de dados paralelamente:

```
vaddps ymm0, ymm1, ymm2
```

AVX Operation

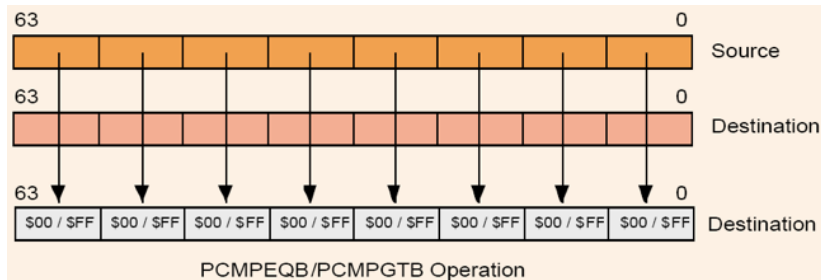


Processamento Vetorial

Comparações

Não modificam CFLAGS:

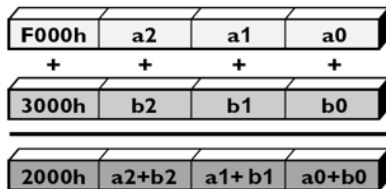
`pcmpeqb mm0, mm1`



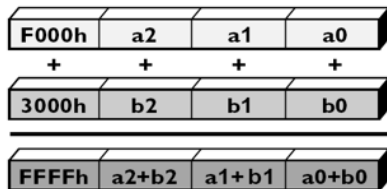
Processamento Vetorial

Aritmética de Saturação

- ▶ Útil em aplicações gráficas
- ▶ Quando uma operação resulta em *under/overflow*
 - ▶ Resultado será o maior/menori inteiro representável
- ▶ Pode ser com ou sem sinal



wrap-around



saturação

Processamento Vetorial

Multiplica e Acumula

pmaddwd xmm0, xmm1

pmaddwd ymm0, ymm1, ymm2

SRC

X3	X2	X1	X0
----	----	----	----

DEST

Y3	Y2	Y1	Y0
----	----	----	----

TEMP

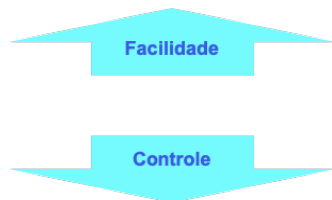
$X3 * Y3$	$X2 * Y2$	$X1 * Y1$	$X0 * Y0$
-----------	-----------	-----------	-----------

DEST

$(X3 * Y3) + (X2 * Y2)$	$(X1 * Y1) + (X0 * Y0)$
-------------------------	-------------------------

Como usar?

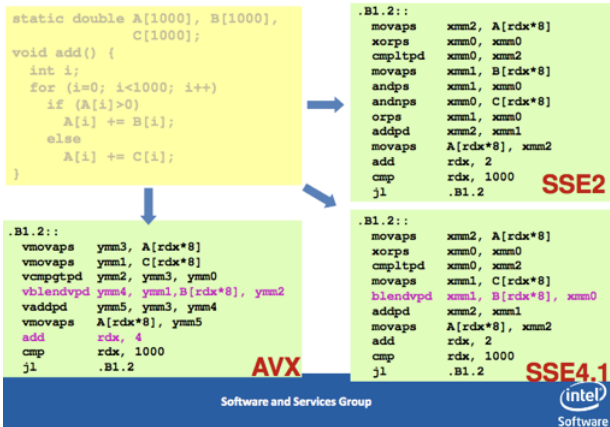
- ▶ Compilador (autovetorização)
- ▶ Bibliotecas (IPP, MKL, etc)
- ▶ Notação específica (Cilk Plus)
- ▶ Compilador (#pragma)
- ▶ Tipos/Funções intrínsecas
- ▶ Assembly¹



¹Exemplo

Como usar?

Autovetorização



```
cc add.c -mavx -S -o add.s
```

Como usar?

Autovetorização

- ▶ Pode encontrar problemas:
 - ▶ Dependências entre as iterações
 - ▶ Tipos de dados misturados
 - ▶ Condição muito complexa
 - ▶ Índice muito complexo
 - ▶ Vetorização ineficiente
 - ▶ Outros...

Como usar?

SIMD/Vector intrinsic



Technologies

- ☐ MMX
- ☐ SSE
- ☐ SSE2
- ☐ SSE3
- ☐ SSSE3
- ☐ SSE4.1
- ☐ SSE4.2
- ☐ AVX
- ☒ AVX2
- ☐ FMA
- ☐ AVX-512
- ☐ KNC
- ☐ SVML
- ☐ Other

Categories

- ☐ Application-Targeted
- ☐ Arithmetic
- ☐ Bit Manipulation
- ☐ Cast
- ☐ Compare
- ☐ Convert
- ☐ Cryptography

maddw



```
__m256i _mm256_madd_epi16 (__m256i a, __m256i b)
```

vpmaddwd

Synopsis

```
__m256i _mm256_madd_epi16 (__m256i a, __m256i b)
#include <immintrin.h>
Instruction: vpmaddwd ymm, ymm, ymm
CPUID Flags: AVX2
```

Description

Multiply packed signed 16-bit integers in **a** and **b**, producing intermediate signed 32-bit integers. Horizontally add adjacent pairs of intermediate 32-bit integers, and pack the results in **dst**.

Operation

```
FOR j := 0 to 7
    i := j*32
    dst[i+31:i] := SignExtend32(a[i+31:i+16]*b[i+31:i+16]) + SignExtend32(a[i+15:i]*b[i+15:i])
ENDFOR
dst[MAX:256] := 0
```

Performance

Architecture	Latency	Throughput (CPI)
Icelake	-	0.5
Skylake	4	0.35
Broadwell	5	1
Haswell	5	1

Como usar?

Compatibilidade

- ▶ Programas vetoriais são muito dependentes do ISA
- ▶ Onde o programa irá rodar?
 - ▶ Sempre na mesma máquina
 - ▶ Em duas máquinas diferentes
 - ▶ Em um cluster com máquinas diversas
 - ▶ Não tenho a menor ideia!
- ▶ CPU dispatch (`cpuid`)
 - ▶ Manual ou automático
 - ▶ Funciona em processadores antigos
 - ▶ Melhor desempenho em processadores novos

Como usar?

Pontos chave para programação

- ▶ Organização de dados eficiente (*data layout*)
- ▶ Eliminação de instruções de desvio (*if-then-else*)

- ▶ Kip R. Irvine, Assembly Language for x86 Processors, 7e, Pearson, 2015
- ▶ **Vetorização SSE & AVX**
- ▶ Intel® Intrinsics Guide
- ▶ x86 and amd64 instruction reference

Programação Paralela: das *threads* aos FPGAs

Processamento Vetorial (SIMD)

Prof. Ricardo Menotti
menotti@ufscar.br

Prof. Maurício Acconcia Dias
macccdias@gmail.com

Prof. Helio Crestana Guardia
helio.guardia@ufscar.br

Departamento de Computação
Universidade Federal de São Carlos

Atualizado em: 7 de maio de 2020