UNIVERSIDADE FEDERAL FLUMINENSE
INSTITUTO DE COMPUTAÇÃO
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Arquiteturas de Computadores – Turma :B1 – Lista 3
Profa.: Simone Martins

1. Explique como funciona um computador vetorial
2. Considere o seguinte código que multiplica dois vetores que contêm valores complexos onde as partes real e imaginária são representadas em precisão dupla:

```
for (i=0;i<300;i++) {
      c_re[i]= a_re[i]*b_re[i] – a_im[i]*b_im[i];
      c_im[i]= a_re[i]*b_im[i] + a_im[i]*b_re[i];
}
```

   a. Gere o código em MIPS para este código e calcule o número de ciclos de relógio que serão necessários para executar este código. Considere que este código será executado em uma máquina com pipeline em 5 estágios, só existe um ciclo de atraso quando há dependência de dados entre uma instrução load e outra instrução e que o desvio pode ser resolvido ao fim do estágio de execução.
   b. Considere que o processador vetorial visto em sala de aula trabalha com um relógio de 700 MHz e possui um tamanho máximo de vetor igual a 64, sendo que cada elemento do vetor possui 64 bits. A unidade load/store tem um overhead de início igual a 15 ciclos, a unidade de multiplicação de 8 ciclos e a unidade de soma/subtração de 5 ciclos.
      b.1. Gere o código deste código em VMIPS. Você deverá dividir o código em blocos para poder carregar os registradores vetoriais com os tamanhos corretos, pois cada vetor do código possui 300 elementos.
      b.2. Considerando que existe chaining e um banco de memória:
         b.2.1. organize o código em convoys e calcule o número de chimes necessários (considere que na primeira iteração a_re e b_re estão carregados na memória)
         b.2.2. calcule o número de ciclos de relógio necessários para obter cada resultado, incluindo o overhead.
      b.3.Calcule o número de ciclos de relógio considerando que existe chaining, três bancos de memória e não existe conflito no acesso à memória

3. Explique como funciona um microprocessador com extensão SIMD
4. Explique como funciona uma GPU
5. Considere uma GPU hipotética com as seguintes características:
   - Freqüência de relógio de 1,5 GHz
   - Contém 16 processadores SIMD, cada um contendo 16 unidades de ponto flutuante de precisão simples
   - Possui uma memória off chip com uma taxa de transmissão igual a 100GB/s.

   Calcule o número máximo de instruções de ponto flutuante por segundo que pode ser atingido por esta GPU sem considerar o acesso à memória. Verifique se a memória especificada pode atender esta taxa de operações ponto em ponto flutuante por segundo.

6. Considere o código em C abaixo:

```
for (i=0; i < 64; i++)
      if (X[i]==0)
            X[i]=Z[i];
      else
            X[i]=X[i]-Y[i];
```

Traduza este código para VMIPS e PTX utilizando as instruções das tabelas abaixo.

Tabela 1: VMIPS

Instruction	Operands	Function
ADDVV.D	V1,V2,V3	Add elements of V2 and V3, then put each result in V1.
ADDVS.D	V1,V2,F0	Add F0 to each element of V2, then put each result in V1.
SUBVV.D	V1,V2,V3	Subtract elements of V3 from V2, then put each result in V1.
SUBVS.D	V1,V2,F0	Subtract F0 from elements of V2, then put each result in V1.
SUBSV.D	V1,F0,V2	Subtract elements of V2 from F0, then put each result in V1.
MULVV.D	V1,V2,V3	Multiply elements of V2 and V3, then put each result in V1.
MULVS.D	V1,V2,F0	Multiply each element of V2 by F0, then put each result in V1.
DIVVV.D	V1,V2,V3	Divide elements of V2 by V3, then put each result in V1.
DIVVS.D	V1,V2,F0	Divide elements of V2 by F0, then put each result in V1.
DIVSV.D	V1,F0,V2	Divide F0 by elements of V2, then put each result in V1.
LV	V1,R1	Load vector register V1 from memory starting at address R1.
SV	R1,V1	Store vector register V1 into memory starting at address R1.
LVWS	V1,(R1,R2)	Load V1 from address at R1 with stride in R2 (i.e., R1 + i × R2).
SVWS	(R1,R2),V1	Store V1 to address at R1 with stride in R2 (i.e., R1 + i × R2).
LVI	V1,(R1+V2)	Load V1 with vector whose elements are at R1 + V2(i) (i.e., V2 is an index).
SVI	(R1+V2),V1	Store V1 to vector whose elements are at R1 + V2(i) (i.e., V2 is an index).
CVI	V1,R1	Create an index vector by storing the values 0, 1 × R1, 2 × R1,...,63 × R1 into V1.
S--VV.D S--VS.D	V1,V2 V1,F0	Compare the elements (EQ, NE, GT, LT, GE, LE) in V1 and V2. If condition is true, put a 1 in the corresponding bit vector; otherwise put 0. Put resulting bit vector in vector-mask register (VM). The instruction S--VS.D performs the same compare but using a scalar value as one operand.
POP	R1,VM	Count the 1s in vector-mask register VM and store count in R1.
CVM		Set the vector-mask register to all 1s.
MTC1 MFC1	VLR,R1 R1,VLR	Move contents of R1 to vector-length register VL. Move the contents of vector-length register VL to R1.
MVTM MVFM	VM,F0 F0,VM	Move contents of F0 to vector-mask register VM. Move contents of vector-mask register VM to F0.

Tabela 2 PTX

Group	Instruction	Example	Meaning	Comments
Arithmetic	arithmetic .type = .s32, .u32, .f32, .s64, .u64, .f64			
	add.type	add.f32 d, a, b	d = a + b;	
	sub.type	sub.f32 d, a, b	d = a – b;	
	mul.type	mul.f32 d, a, b	d = a * b;	
	mad.type	mad.f32 d, a, b, c	d = a * b + c;	multiply-add
	div.type	div.f32 d, a, b	d = a / b;	multiple microinstructions
	rem.type	rem.u32 d, a, b	d = a % b;	integer remainder
	abs.type	abs.f32 d, a	d = |a|;	
	neg.type	neg.f32 d, a	d = 0 - a;	
	min.type	min.f32 d, a, b	d = (a < b)? a:b;	floating selects non-NaN
	max.type	max.f32 d, a, b	d = (a > b)? a:b;	floating selects non-NaN
	setp.cmp.type	setp.lt.f32 p, a, b	p = (a < b);	compare and set predicate
	numeric .cmp = eq, ne, lt, le, gt, ge; unordered cmp = equ, neu, ltu, leu, gtu, geu, num, nan			
	mov.type	mov.b32 d, a	d = a;	move
	selp.type	selp.f32 d, a, b, p	d = p? a: b;	select with predicate
	cvt.dtype.atype	cvt.f32.s32 d, a	d = convert(a);	convert atype to dtype
Special Function	special .type = .f32 (some .f64)			
	rcp.type	rcp.f32 d, a	d = 1/a;	reciprocal
	sqrt.type	sqrt.f32 d, a	d = sqrt(a);	square root
	rsqrt.type	rsqrt.f32 d, a	d = 1/sqrt(a);	reciprocal square root
	sin.type	sin.f32 d, a	d = sin(a);	sine
	cos.type	cos.f32 d, a	d = cos(a);	cosine
	lg2.type	lg2.f32 d, a	d = log(a)/log(2)	binary logarithm
	ex2.type	ex2.f32 d, a	d = 2 ** a;	binary exponential
Logical	logic.type = .pred,.b32, .b64			
	and.type	and.b32 d, a, b	d = a & b;	
	or.type	or.b32 d, a, b	d = a | b;	
	xor.type	xor.b32 d, a, b	d = a ^ b;	
	not.type	not.b32 d, a, b	d = ~a;	one's complement
	cnot.type	cnot.b32 d, a, b	d = (a==0)? 1:0;	C logical not
	shl.type	shl.b32 d, a, b	d = a << b;	shift left
	shr.type	shr.s32 d, a, b	d = a >> b;	shift right
Memory Access	memory.space = .global, .shared, .local, .const; .type = .b8, .u8, .s8, .b16, .b32, .b64			
	ld.space.type	ld.global.b32 d, [a+off]	d = *(a+off);	load from memory space
	st.space.type	st.shared.b32 [d+off], a	*(d+off) = a;	store to memory space
	tex.nd.dtyp.btype	tex.2d.v4.f32.f32 d, a, b	d = tex2d(a, b);	texture lookup
	atom.spc.op.type	atom.global.add.u32 d,[a], b atom.global.cas.b32 d,[a], b,	atomic { d = *a; *a = cop(*a, b); }	atomic read-modify-write operation
	atom.op = and, or, xor, add, min, max, exch, cas; .spc = .global; .type = .b32			
Control Flow	branch	@p bra target	if (p) goto target;	conditional branch
	call	call (ret), func, (params)	ret = func(params);	call function
	ret	ret	return;	return from function call
	bar.sync	bar.sync d	wait for threads	barrier synchronization
	exit	exit	exit;	terminate thread execution

7. Neste exercício, você deverá examinar vários laços e verificar seu potencial para ser paralelizado:

7.1. Verifique se o código abaixo possui dependência de laço

```
for (i=0; i<100;i++) {
A[i]=B[2*i+4];
B[4*i+5]=A[i];
}
```

7.2. No laço abaixo, encontre todas as dependências verdadeiras, dependências de saída, e anti-dependências. Elimine as dependências e anti-dependências por renomeação.

```
for (i=0; i<100;i++) {
      A[i]=A[i]*B[i];
      B[i]=A[i]+c;
      A[i]=C[i]*c;
      C[i]=D[i]*A[i];
}
```

7.3. Considere o laço abaixo:

```
for (i=0; i<100;i++) {
      A[i]=A[i]+B[i]; /* S1 */
      B[i+1]=C[i]+D[i]; /* S2 */

}
```

Existem dependências entre S1 e S2? Este laço é paralelizável? Se não é, mostre como torná-lo paralelizável.