

Arquitetura e Organização de Computadores II

Arquiteturas vetoriais

Prof. Nilton Luiz Queiroz Jr.

Arquiteturas SIMD

- Arquiteturas SIMD podem explorar paralelismo em nível de dados para diversas aplicações como:
 - Cálculos matriciais;
 - Processamento de imagens;
 - Processamento de sons;
- Em geral são as arquiteturas SIMD são mais eficientes que a MIMD em termos de energia;
 - Só é necessário a busca de uma instrução para efetuar cálculos sobre diversos dados;



Arquiteturas SIMD

- Existem diversas variantes das arquiteturas SIMD:
 - Arquiteturas Vetoriais;
 - Extensões de conjuntos de instruções SIMD para multimídia;
 - GPUs;
 - Entre outras;



Arquiteturas SIMD

- As arquiteturas vetoriais são mais antigas que as extensões SIMD e as GPUs;
- Mais fáceis de compilar e compreender o funcionamento;
 - São basicamente a execução em pipeline de diversas operações de dados;
- Consideradas muito caras para microprocessadores até bem recentemente;
 - Parte desse custo eram os transistores;
 - Outra parte o custo de largura de banda suficiente para a DRAM;



Arquiteturas SIMD

- As variações SIMD para multimídia são encontradas na maior parte das de conjuntos de instruções que dão suporte a multimídia nos dias atuais;
- Na arquitetura x86 :
 - As instruções MMX (Multimídia Extensions);
 - Instruções SSE (Streaming SIMD Extensions);
 - Instruções AVX (Advanced Vector Extensions);



Arquiteturas SIMD

- GPUs (Graphics Processor Units) oferecem desempenho computacional maior que os computadores multicore de hoje;
- Tem características semelhantes às arquiteturas vetoriais e também características próprias;



Arquiteturas Vetoriais

- **Arquiteturas vetoriais:**
 - Coletam conjuntos de dados da memória para arquivos de registradores sequenciais
 - Operam os dados desses arquivos de registradores; e
 - Escrevem os resultados de volta na memória ;
- **Uma única instrução opera sobre vetores;**
- Esses arquivos de registradores funcionam como buffers;
 - Pode ser usado para ocultar latência de memória ou aproveitar largura de banda;
- **Loads e Stores em arquiteturas vetoriais tem pipeline profundo;**
 - O custo é amortizado, como se fosse pagar o delay alto apenas por um dos diversos loads ou stores;

Arquiteturas Vetoriais

- Arquitetura Vetorial VMIPS:
- A arquitetura vetorial VMPIS é composta principalmente por:
 - Registradores vetoriais;
 - Unidades funcionais vetoriais;
 - Unidades de load/store vetoriais (unidade de memória vetorial);
 - Conjunto de registradores escalares;

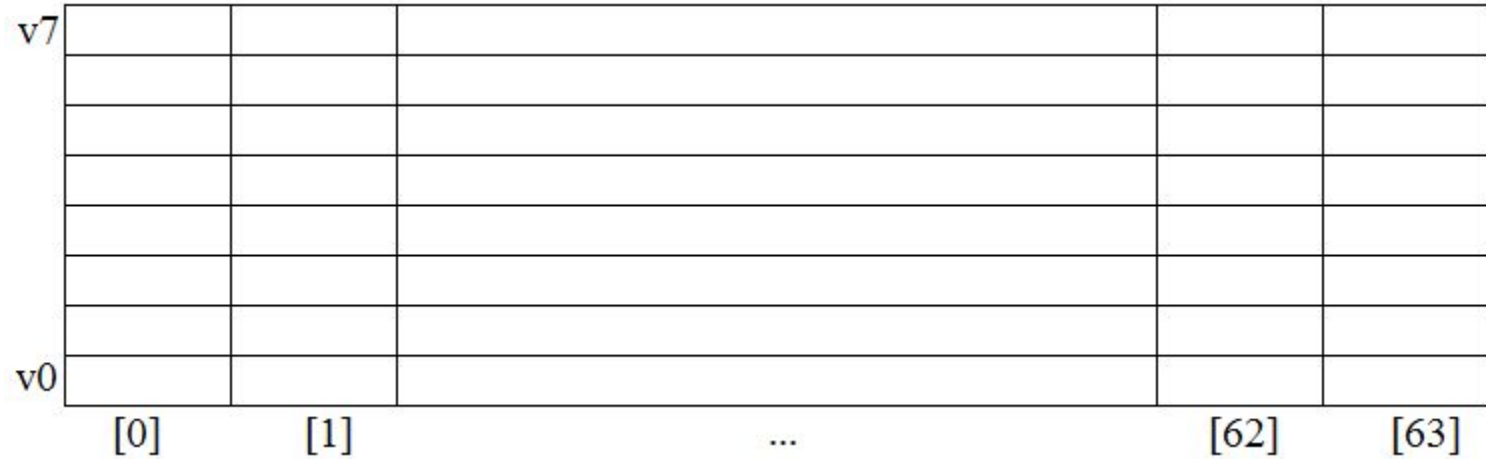


Registradores Vetoriais

- Cada registrador vetorial é um banco de tamanho fixo;
- Mantém armazenados vetores;
- A arquitetura VMPIS possui 8 registradores vetoriais, cada um armazenando 64 valores;
- O banco de registradores possui portas para alimentar todas as unidades funcionais;



Registradores Vetoriais



Unidades funcionais vetoriais

- Cada unidade funcional vetorial tem um pipeline e tem capacidade de iniciar uma operação por ciclo;
- É necessária uma unidade de detecção de hazards;
 - Hazards de dados;
 - Hazards de unidades funcionais;



Unidades de Load/Store vetoriais

- A unidade de memória vetorial é responsável por carregar ou armazenar um vetor na memória principal;
- Na arquitetura VMIPS essa unidade faz uso de pipeline;
 - Isso torna possível um valor movido entre memória e registrador após a latência inicial;



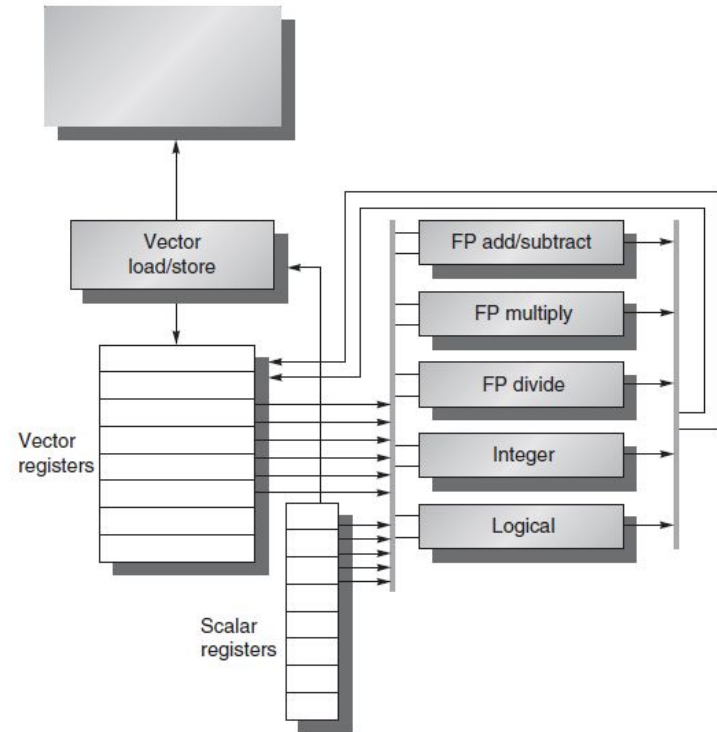
Conjunto de registradores escalares

- Registradores normais de propósito geral;
- Podem prover dados para as unidades vetoriais;
 - Algumas instruções vetoriais podem ser feitas com operandos escalares;
 - Soma de um escalar com os valores do vetor;
 - Multiplicação de um escalar com os valores do vetor;
 - Entre outras;



Arquitetura VMIPS

- Parte de ponto flutuante da arquitetura VMIPS



Arquiteturas Vetoriais

- Para a realização de operações sobre vetores existem instruções específicas;
 - Na arquitetura VMIPS algumas delas são:
 - ADDVV.D e ADDVS.D;
 - Soma de vetor com vetor e soma de vetor com escalar
 - SUBVV.D, SUBVS.D, SUBSV.D
 - Subtração de vetor com vetor e subtração de escalar com vetor;
 - LV e SV;
 - Load e store para vetores
 - etc;



Exemplo de código

- Código em C

```
for (i=0; i<64; i++)  
    C[i] = A[i] * B[i];
```

- Código em Assembly
escalar

```
        LI R4, 64  
loop:   L.D F0, 0(R1)  
        L.D F2, 0(R2)  
        MUL.D F4, F2, F0  
        S.D F4, 0(R3)  
        DADDIU R1, 8  
        DADDIU R2, 8  
        DADDIU R3, 8  
        DSUBIU R4, 1  
        BNEZ R4, loop
```

- Código em Assembly
vetorial

```
        LI VLR, 64  
        LV V1, R1  
        LV V2, R2  
        MULVV.D V3, V1, V2  
        SV V3, R3
```



Tamanho de vetor

- Em um processador como o VMIPS o tamanho de vetor é de 64;
- Suponha que se deseja operar apenas 4 elemento no vetor, como poderia ser feito isso?



Tamanho de vetor

- Em um processador como o VMIPS o tamanho de vetor é de 64;
- Suponha que se deseja operar apenas 4 elemento no vetor, como poderia ser feito isso?
 - Criar um registrador para controlar o tamanho de vetor
 - VLR;
 - Esse registrador deve ser sempre menor ou igual ao tamanho máximo de vetor;
 - Algumas arquiteturas facilitam isso com um parâmetro, o MVL, que representa o tamanho máximo do vetor, assim, alterar o tamanho máximo de vetor implica apenas em alterar o MVL ;
 - Outras arquiteturas, necessitam de alterações no conjunto de instruções ao alterar o tamanho máximo de vetor;

Exemplo de execução

```
for (i=0; i<4; i++)
    c[i] = a[i]*x;
```

ADDI	R9, ZERO, 4	;
MTC1	VLR, R9	;move o conteudo de R9 para VLR
LV	V1, R1	;Carrega 4 valores, iniciando no endereço R1
MULVS.D	V2,V1,F0	;faz a multiplicação do vetor v1 por f0 e armzena em v2
SV	V2,R2	;armzena 4 valores, iniciando no endereço r2

LV	V2	R2		F	D	RO	L0	L1	W																			
							RO	L0	L1	W																		
								RO	L0	L1	W																	
									RO	L0	L1	W																
MULVS.D	V3	V1	F2		F	D	D	D	D	D	D	D	RO	M0	M1	M2	M3	W										
													RO	M0	M1	M2	M3	W										
														RO	M0	M1	M2	M3	W									
															RO	M0	M1	M2	M3	W								
SV	V3	R3			F	F	F	F	F	F	F	D	D	D	D	D	D	D	D	D	R	S0	S1	W				
																						RO	S0	S1	W			
																							RO	S0	S1	W		
																								RO	S0	S1	W	

Tamanho de vetor

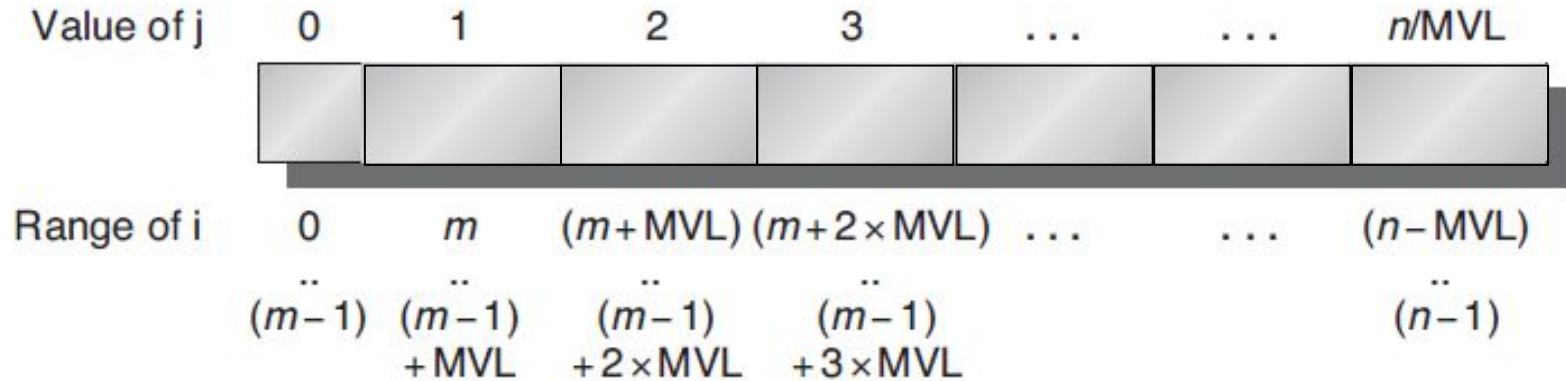
- Quando o tamanho de vetor não for conhecido durante a execução é necessário tratar casos que ele possa ultrapassar o tamanho total;
- Para resolver esses problemas usa-se uma técnica chamada de strip mining;
 - Cria-se código de um jeito que cada operação vetorial seja menor ou igual ao MVL

```
low =0;
vl = n%MVL
for (j=0; j < n/MVL; j++){
    for(i=low; i<low+VL; i++){
        c[i]=a[i]*x;
    }
    low+=vl;
    vl=MVL;
}
```

//encontra-se o resto das operações
//loop externo
//loop interno, que será convertido em código vetorial
//operação vetorial

//incremento o endereço de início do vetor
//após a primeira iteração todos valores
//de incremento devem ser o tamanho do vetor

Strip Mining



Arquiteturas Vetoriais

- O tempo de execução para operações vetoriais depende principalmente de 3 fatores:
 - Tamanhos dos vetores do operando;
 - Hazards estruturais;
 - Dependência de dados;
- Instruções que podem ser inicializadas em um mesmo ciclo são chamadas de comboio;
 - Instruções em um mesmo comboio não podem ter hazards;
 - De dados;
 - De estrutura;



Arquiteturas vetoriais

- Uma fonte de overhead em arquiteturas vetoriais é o tempo de início (start-up time) do vetor;
 - O tempo de início do vetor é o intervalo entre o início da execução da instrução e a saída do resultado da primeira operação do comboio de instruções
 - Depende da latência do pipeline da operação vetorial;
 - Quanto mais profundo o pipeline maior será o tempo de início;

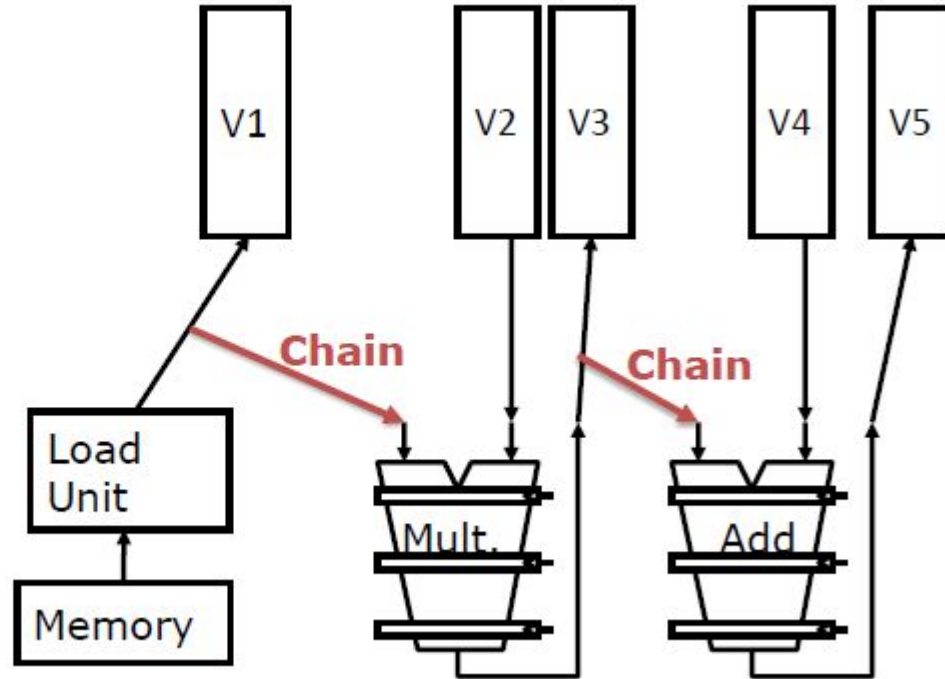


Encadeamento

- Uma maneira de melhorar o desempenho da execução é usar encadeamento (chaining);
- O encadeamento permite uma operação começar quando os elementos do operando fonte estiverem prontos
 - Não é necessário esperar pelo término da instrução que está produzindo esses valores;
- O encadeamento só ocorre de uma unidade funcional para outra;
 - Por exemplo, a unidade de load, ao terminar de obter o valor da memória pode repassá-lo para a unidade que necessita dele
 - Não é possível fazer encadeamento quando existem hazards estruturais;
- Pode ser feito pelo banco de registradores ou por um adiantamento
 - Quando feito por adiantamento se torna interessante mais portas para leitura e escrita;
 - Para evitar hazards estruturais;

Encadeamento

LV V1
MULVV V3, V1, v2
ADDVV V5, V3, v4



Encadeamento

- Sem encadeamento, em casos de dependência, é necessário esperar o resultado ser escrito para que outra unidade funcional possa começar a processar;

LV V1
MULVV V3, V1, v2
ADDVV V5, V3, v4



Encadeamento

- Com encadeamento é possível começar a execução assim que o primeiro resultado já apareceu;

LV V1
MULVV V3,V1,v2
ADDVV V5,V3, v4



Encadeamento

- A execução das instruções com encadeamento

```
ADDI    R9, ZERO, 4
```

MTC1 VLR, R9

LV V1, R1

MULVS.D V2,V1,F0

SV V2,R2

LV	V2	R2		F	D	RO	L0	L1	W													
							RO	L0	L1	W												
								RO	L0	L1	W											
									RO	L0	L1	W										
MULVS.D	V3	V1	F2		F	D	D	D	D	RO	M0	M1	M2	M3	W							
											RO	M0	M1	M2	M3	W						
												RO	M0	M1	M2	M3	W					
													RO	M0	M1	M2	M3	W				
SV	V3	R3			F	F	F	F	D	D	D	D	D	D	RO	S0	S1	W				
																RO	S0	S1	W			
																	RO	S0	S1	W		
																		RO	S0	S1	W	

Encadeamento

- A execução das instruções com encadeamento por adiantamento (forwarding) com mais portas de leitura e escrita nos registradores:

```

ADDI      R9, ZERO, 4
MTC1      VLR, R9
LV         V1, R1
MULVS.D   V2,V1,F0
SV         V2,R2
    
```

LV	V2	R2		F	D	RO	L0	L1	W									
							RO	L0	L1	W								
								RO	L0	L1	W							
									RO	L0	L1	W						
MULVS.D	V3	V1	F2		F	D	D	RO	M0	M1	M2	M3	W					
									RO	M0	M1	M2	M3	W				
										RO	M0	M1	M2	M3	W			
											RO	M0	M1	M2	M3	W		
SV	V3	R3			F	F	D	D	D	D	RO	S0	S1	W				
												RO	S0	S1	W			
													RO	S0	S1	W		
														RO	S0	S1	W	

Arquiteturas vetoriais

- Arquiteturas vetoriais permitem que grande parte do paralelismo seja passado para o hardware;
- Uma única instrução vetorial inclui dezenas (ou centenas) de operações independentes;
- Devido sua semântica paralela a instrução pode ser executada em:
 - Somente uma unidade funcional;
 - Um array de unidades paralelas;
 - Combinação de unidades paralelas em pipeline;
- Na arquitetura VMIPS um elemento N de um registrador vetorial se relacione com o elemento N de outro registrador vetorial;



Múltiplas pistas

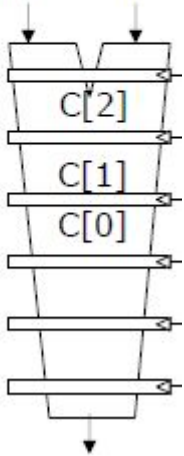
- A ideia de usar múltiplas pistas (lanes) é fazer com que exista diversas unidades funcionais, onde cada uma delas irá ser responsável por um conjunto de registradores;
 - Reduzir a quantidade de operações que cada uma das unidades funcionais irá realizar em uma instrução vetorial;
- Não existe comunicação entre essas pistas;
- Adicionar múltiplas pistas:
 - Permite redução de clock, voltagem e energia sem perder desempenho de pico;
 - Dobrar o número de pistas e reduzir pela metade o clock mantém o mesmo desempenho potencial;
 - Exige pouco aumento na complexidade de controle;
 - Não exige alterações no código de máquina;



Uma pista x Múltiplas pistas

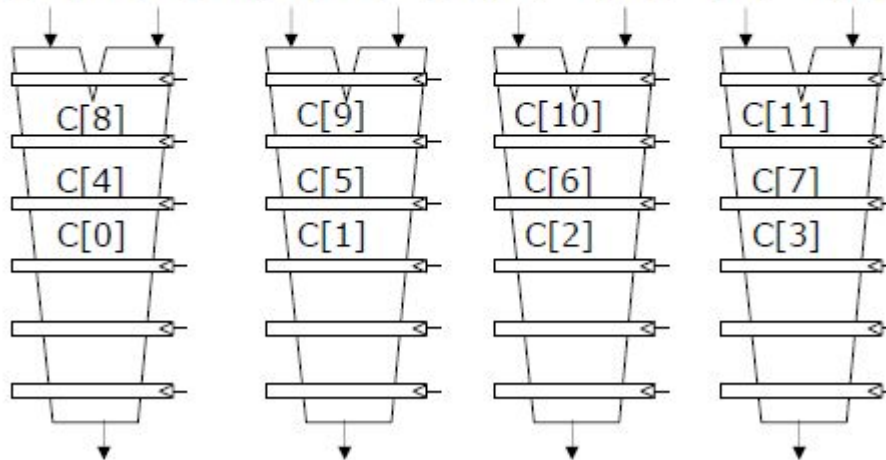
Uma pista

A[6] B[6]
A[5] B[5]
A[4] B[4]
A[3] B[3]

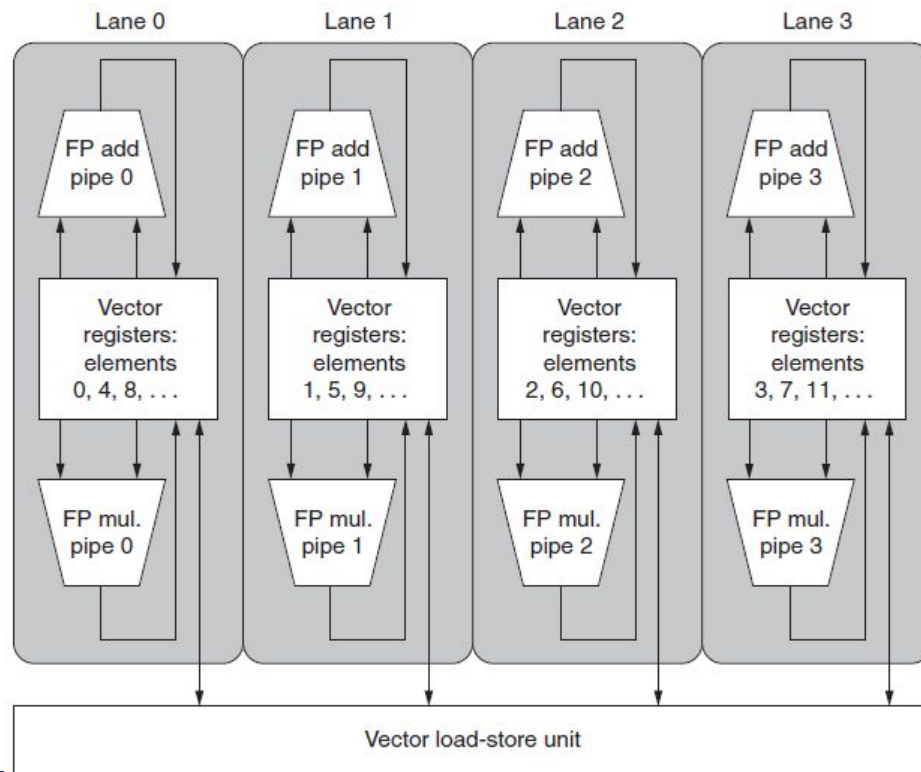


Múltiplas Pistas

A[24] B[24] A[25] B[25] A[26] B[26] A[27] B[27]
A[20] B[20] A[21] B[21] A[22] B[22] A[23] B[23]
A[16] B[16] A[17] B[17] A[18] B[18] A[19] B[19]
A[12] B[12] A[13] B[13] A[14] B[14] A[15] B[15]



Múltiplas Pistas



Múltiplas pistas

- Execução das instruções com duas lanes:

ADDI	R9, ZERO, 4
MTC1	VLR, R9
LV	V1, R1
MULVS.D	V2,V1,F0
SV	V2,R2

LV	V2	R2		F	D	RO	L0	L1	W								
						RO	L0	L1	W								
							RO	L0	L1	W							
							RO	L0	L1	W							
MULVS.D	V3	V1	F2		F	D	D	RO	M0	M1	M2	M3	W				
								RO	M0	M1	M2	M3	W				
									RO	M0	M1	M2	M3	W			
									RO	M0	M1	M2	M3	W			
SV	V3	R3			F	F	D	D	D	D	RO	S0	S1	W			
												RO	S0	S1	W		
													RO	S0	S1	W	
														RO	S0	S1	W

Desvios em arquiteturas vetoriais

- Programas que contém desvios condicionais dentro dos laços não podem ser executados no modo vetorial;
 - Instruções **if-then**, por exemplo, introduzem dependências de controle no laço;
- Suponha o seguinte trecho de código:

```
for (i=0; i<64;i++){  
    if (x[i] > 0){  
        x[i]=x[i]+y[i]  
    }  
}
```

- O que poderia ser feito para que se torne possível executá-lo no modo vetorial?

Desvios em arquiteturas vetoriais

- Para solucionar problemas dessa natureza se utiliza uma extensão de controle de máscara vetorial;
 - Usam-se registradores de máscara;
- Esses registradores fornecem a execução condicional de cada operação de elemento de uma instrução vetorial;
- Em geral esses registradores são uma espécie de vetor de valores booleanos;
- Quando o registrador de máscara vetorial é habilitado as instruções vetoriais operam somente sobre os elementos vetoriais cujo a máscara é 1;



Registradores de máscara

- Esses registradores são habilitados com instruções para a arquitetura:
- No caso da arquitetura VMIPS essas instruções são:
 - S - - VV.D V1,V2;
 - Comparação de V1[i] com V2[i]
 - S - - VS.D V1,F0;
 - Comparação de todos elementos de V1 com uma única constante;
 - Em ambas instruções os valores - - são substituídos pela operação relacional que será feita;
 - SEQVV.D ou SEQVS.D: seta 1 quando a condição de igualdade entre os dois operandos é satisfeita;
 - SNEVV.D ou SNEVS.D: seta 1 quando a condição de desigualdade entre os dois operandos é satisfeita;
 - SGTVV.D ou SGTVS.D: seta 1 quando cada elemento do primeiro operando é maior que o do segundo;
 - etc...

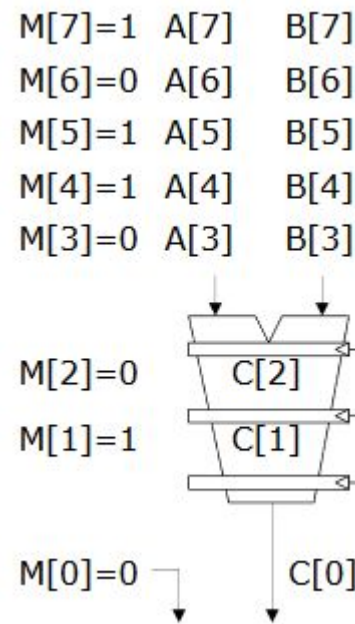
Desvios em arquiteturas vetoriais

- Assim, com essas instruções e o vetor de máscara (VM), é possível fazer execução no modo vetorial do loop em C;

for (i=0; i<64;i++){	LV	V1,Rx	;faz load do vetor X em V1
if (x[i] > 0){	LV	V2,Ry	;faz load do vetor Y em V2
x[i]=x[i]+y[i]	L.D	F0,#0	;faz load do valor 0 em V0
}	SGTVS.D	V1,F0	;seta o VM[i] to 1 if V1[i] > F0
}	ADDVV.D	V1,V1,V2	;soma os vetores V1 e V2
	SV	V1,Rx	;armazena o vetor V1 em X

Desvios em arquiteturas vetoriais

- A máscara vetorial causa overhead na execução;
 - As instruções que não deveriam ser executadas são executadas, porém os registradores destinos não sofrem a escrita;



Desvios em Arquiteturas vetoriais

- Mesmo com esse overhead a execução ainda vale a pena a execução vetorial;
 - Eliminação de diversos desvios;
 - Redução no número de buscas e decodificações de instruções;



Unidades de Load/store vetoriais

- As unidades de load/store vetoriais são mais complexas que as unidades funcionais aritméticas;
- Podem ocorrer stalls no banco de memória o que reduziria o throughput;
- Para manter a taxa de palavras buscadas ou armazenadas de uma por clock o sistema de memória deve ser capaz de produzir, ou aceitar, essa quantidade de dados;
 - Normalmente, para que o sistema tenha essa capacidade espalha-se os acessos por vários bancos de memória;



Unidades de Load/store vetoriais

- Existem 3 principais motivos para o uso de diversos bancos de memória:
 - Para dar suporte a múltiplos acessos a memória simultaneamente o sistema de memória deve ter diversos bancos e ser capaz de controlar os endereços para os bancos de forma independente, pois o tempo de ciclo de banco de memória é muito maior que o de CPU;
 - Muitos processadores vetoriais permitem que sejam carregadas palavras que não estão em sequência na memória, assim precisa-se de endereçamento independente do banco;
 - Muitos computadores vetoriais admitem múltiplos processadores compartilhando o mesmo sistema de memória, assim cada processador irá gerar seu fluxo independente de endereços;
- Em combinação, esses recursos levam a um grande número de bancos de memória independentes;



Unidades de Load/store vetoriais

- Exemplo:
 - O processador Cray T90 possui 32 processadores, cada qual é capaz de gerar 4 carregamentos e dois armazenamentos por ciclo. O tempo de clock dos processadores é de 2,167 ns e o tempo de clock dos bancos de memória é de 15ns. Qual o número mínimo de bancos de memória para permitir que todas as CPUs executem na largura de banda total de memória?



Exemplo

- A quantidade de acessos aos bancos de memória que cada processador faz é no máximo de 2 stores e 4 loads por ciclo. Temos assim um total de 6 acessos a memória por ciclo para cada processador;
- Assim, temos que o número de referências a memória a cada ciclo é de $32 \times 6 = 192$;
- A quantidade de ciclos que cada banco fica ocupado é dada por $15/2,167 = 6,92$, que podemos arredondar para 7 ciclos.
- Dessa forma, a quantidade de bancos é $192 \times 7 = 1344$ bancos;



Stride

- Algumas vezes é necessário fazer o acesso de elementos que não estão próximos na memória;
 - Por exemplo, no código abaixo, a multiplicação das linhas de A com as colunas de B poderia ser vetorizada;

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

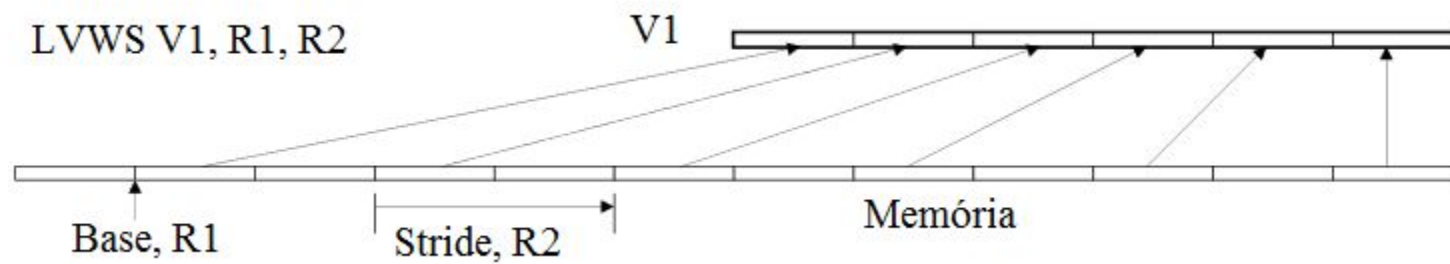
- Porém as colunas de B não estão em sequência na memória;

Stride

- Para processadores vetoriais que não possuem cache é necessário o uso de uma técnica para buscar elementos de um vetor que não adjacentes na memória;
- A distância entre dois elementos de um vetor, que pode ser armazenada em um registrador, é chamada de passo;
 - No exemplo, o passo para armazenar as colunas de B na memória é de 100 palavras duplas (800 bytes);
- Para fazer esse carregamento e armazenamento desses casos é necessário o uso de instruções que permitam passos;
 - LVWS (Load Vector With Strides);
 - SVWS (Store Vector With Strides);



Stride



Gather-Scatter

- Outro problema comum são as matrizes dispersas;
- Uma técnica para matrizes dispersa é geralmente acessar os elementos de alguma forma compactada e depois fazer acesso indireto a eles;
- Por exemplo:
 - Somar os valores de dois vetores dispersos A e C, onde as posições dos valores diferentes de 0 são armazenados em vetores de índices k e m.

```
for(i=0;i<n;i++){  
    A[k[i]]=A[k[i]]+C[m[i]]  
}
```



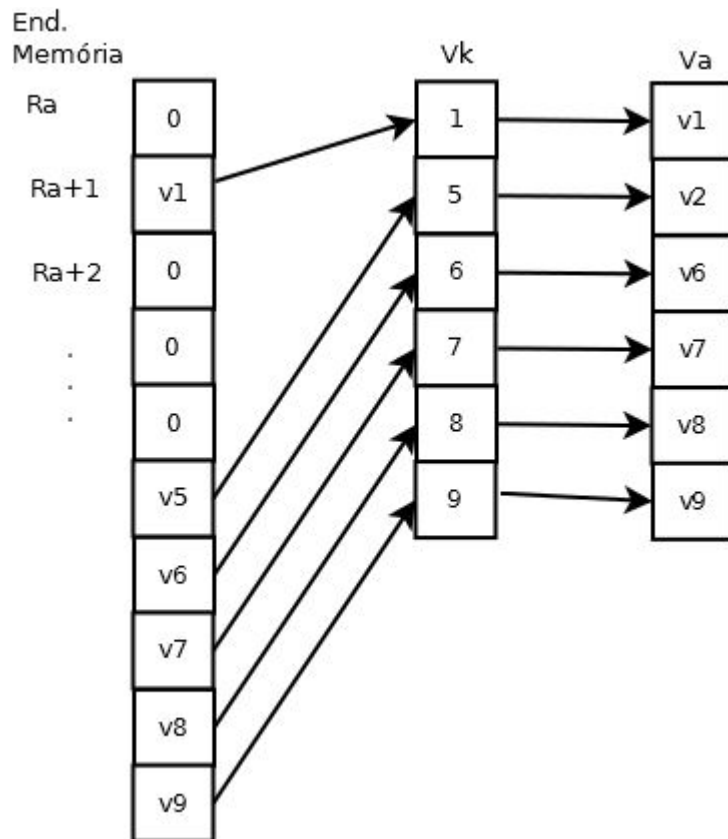
Gather-Scatter

- O principal suporte que uma arquitetura vetorial costuma oferecer para esse tipo de situação são as operações Gather-Scatter;



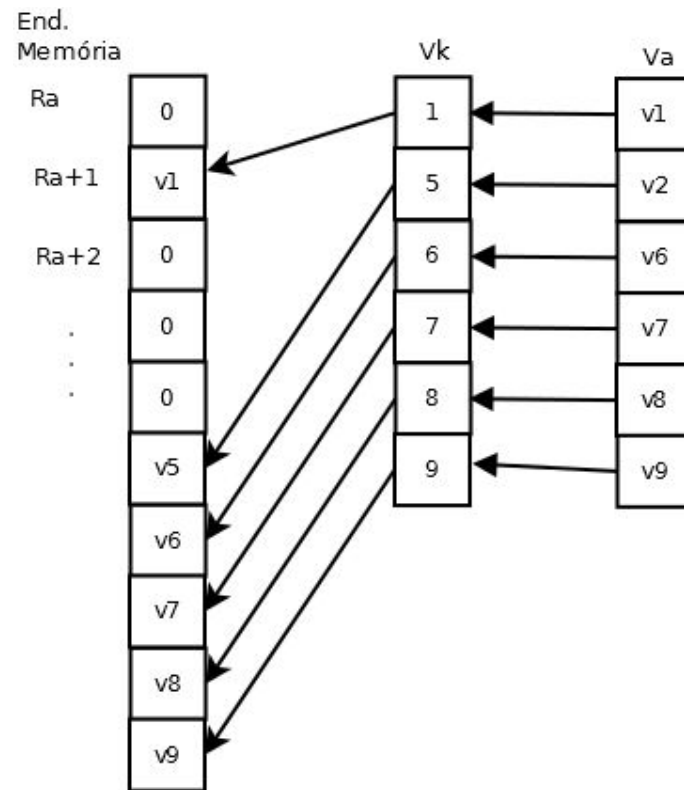
Gather

- Gather: Recebe um vetor para armazenar e um vetor de índice. Os elementos buscados são os elementos que estão no vetor de índice somados ao endereço base;
- Exemplo:
 - LVI Va, (Ra+Vk);
- Em outras palavras para cada elemento do registrador vetorial:
 - $Va[i] = Memória[Ra+Vk[i]]$



Scatter

- Semelhante a operação gather, porém ao invés de buscar na memória e armazenar no registrador, faz o contrário, copia o registrador para o endereço de memória indexado pela soma do registrador base com o registrador vetorial;
 - Exemplo:
 - $SVI(Ra+Vk), Va$
- Em outras palavras para cada elemento do registrador vetorial:
 - $Memória[Ra+Vk[i]] = Va[i]$



Gather-Scatter

- Código em C:

```
for(i=0;i<n;i++){  
    A[k[i]]=A[k[i]]+C[m[i]]  
}
```

- Instruções vetoriais:

LV	Vk, Rk	;load K
LVI	Va, (Ra+Vk)	;load A[K[]]
LV	Vm, Rm	;load M
LVI	Vc, (Rc+Vm)	;load C[M[]]
ADDVV.D	Va, Va, Vc	;soma os vetores
SVI	(Ra+Vk), Va	;store A[K[]]

Gather-Scatter

- Um compilador não saberia que os valores de k são valores distintos, logo não conseguiria prever se existe ou não dependência;
 - Programadores podem usar diretivas para execução em modo vetorial para auxiliar no processo de compilação;




Arquiteturas vetoriais

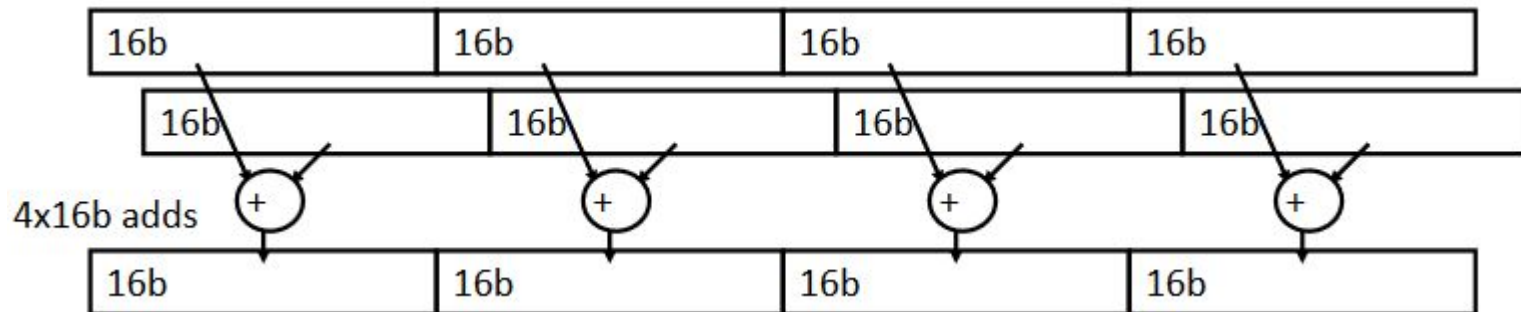
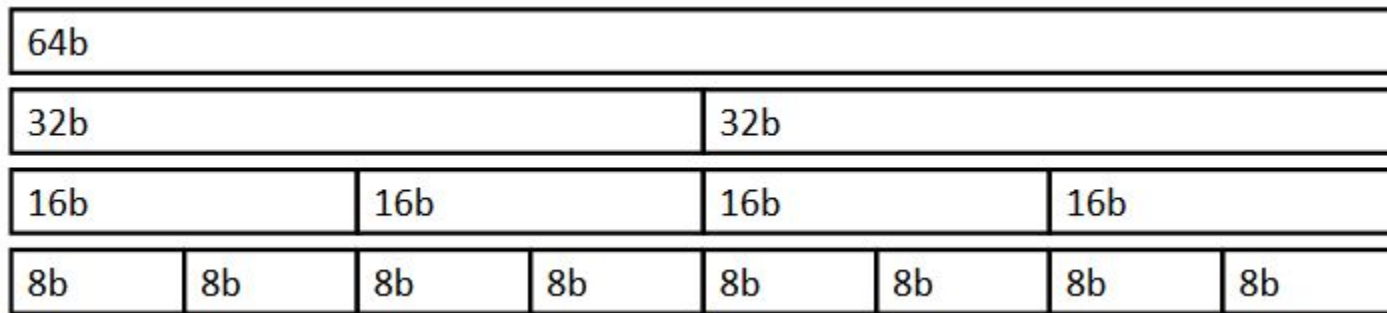
- Quando se programa em arquiteturas vetoriais pode existir comunicação entre programador e compilador
 - O compilador pode detectar algumas partes de código vetorial, informar ao programador que tal trecho será vetorizado;
 - O programador pode informar ao compilador, com o uso de diretivas que tal trecho deve ser vetorizado;
- Dependências reais de dados entre os laços devem ser reestruturadas;
 - Novos algoritmos ou técnicas de implementação para problemas;
 - Exemplo:
 - Para somar todos os dados de um vetor pode-se dividir o vetor no meio e somar dois vetores, e se repetir esse processo até que reste uma quantidade de elementos que compense somar de maneira “sequencial”

Extensões SIMD para Multimídia

Extensões SIMD para multimídia

- Observou-se que aplicações multimídia operam com tipos de dados mais restritos que aqueles para os quais processadores x86 foram otimizados;
 - 8 bits para as 3 cores primárias;
 - 8 ou 16 bits para amostras de áudio;
 - Desse modo um processador poderia, com um somador de 64 bits, um processador poderia realizar várias operações sobre esses pequenos elementos;
 - Basta particionar o carry, o que tem um custo baixo;
 - Desse modo, a ideia dessas instruções é dividir um registrador e trata-los como vetor
- 

Extensões SIMD para multimídia



Extensões SIMD para multimídia

- As implementações MMX usam os registradores que compõem a pila de ponto flutuante;
 - Os registradores de ponto flutuante de 64 bits foram reusados para instruções MMX;
 - Os registradores da arquitetura x86 ficam em uma pilha a parte;
- As implementações SSE tem registradores separados de 128 bits;
 - Nas extensões SSE foi possível operar sobre ponto flutuante;
 - SSE operações com PF de precisão simples (32 bits - 4 operações);
 - SSE2 operações com PF de precisão simples ou dupla (64 bits - 2 operações);
 - Os registradores SSE poderiam ser usados como registradores PF;
- As implementações AVX tem registradores de 256 bits;
 - A AVX foi implementada de maneira que já tem preparações prontas para estender os registradores para 512 e 1024 bits



Extensões SIMD vs Arquiteturas Vetoriais

- Extensões multimídia tem um conjunto de instruções limitado;
 - Não existe registrador para controlar o tamanho do vetor;
 - Sem gather/scatter;
- São fáceis de adicionar a unidade aritmética padrão e de implementar;
- Requerem uma quantidade menor de largura de banda;
- Não precisa lidar com problemas de memória virtual;
 - Por exemplo: falha de página no meio do vetor;
- Como os vetores são pequenos, se torna fácil introduzir instruções que podem ajudar com novos padrões de mídia;



Referências

HENNESSY, John L.;PATTERSON, David A. Computer architecture: a quantitative approach. Elsevier, 2011.

Wentzlaff, David. Vector, SIMD, and GPUs. Notas de Aula.

