CAP-387(2016) – Tópicos Especiais em Computação Aplicada: Construção de Aplicações Massivamente Paralelas

Aula 17: Avaliação de Vetorização

Celso L. Mendes, Stephan Stephany

LAC / INPE

Emails: celso.mendes@inpe.br, stephan.stephany@inpe.br

Benchmark de Vetorização

- Benchmark Sintético: TSVC
  - Compilação com e sem vetorização dos 151 loops
  - Execução das duas versões, cálculo do speedup: $S=T_{no\text{-}vec}/T_{vec}$
  - Mudanças manuais em alguns dos loops não-vetorizados
  - Definições para os loops:
    - ***Auto-vectorized***: vetorizado pelo compilador, S ≥ 1.15
    - ***Perfectly auto-vectorized***: vetorizado pelo compilador e não pode ser mais otimizado manualmente
    - ***Not Vectorizable***: Nem pelo compilador nem manualmente é possível obter S ≥ 1.15

Benchmark de Vetorização (cont.)

- Principais razões para insucesso na vetorização:
  - Falta de análise precisa pelo compilador
  - Necessidade de <u>transformações</u> nos códigos fonte
    - Pode ser auxiliada pelo programador experiente
  - Necessidade de um modelo de ganho conhecido pelo compilador:
    - Compilador deve avaliar se certa transformação é lucrativa
    - Em alguns casos, compilador pode gerar duas versões do loop, e fazer a escolha da mais eficiente em tempo de execução

Benchmark de Vetorização (cont.)

- Fatores importantes para vetorização:
    - Acessos com *stride* unitário: única forma suportada na maioria dos processadores
        - Porém, muitos programas reais têm *stride* > 1 !
        - Elementos podem ser acessados individualmente, antes de uma operação vetorial; mas há overhead

    Exemplos:
    - acesso a uma linha de uma matriz 2D, em Fortran
    - acesso a uma coluna de uma matriz 2D, em C

Benchmark de Vetorização (cont.)

- Fatores importantes para vetorização (cont.)

  Exemplo-2: array de estruturas

```
typedef struct {int x,y,z} point;
point pt[LEN];
for (int i=0; i<LEN; i++)
   pt[i].x *= scale;
```

Código Transformado:

```
int ptx[LEN],int pty[LEN],int ptz[LEN];
for (int i=0; i<LEN; i++)
    ptx[i] *= scale;
```

→ vetorizável!

Porém, nova estrutura deve ser refletida em todo o programa!

Exemplo-3: outra forma com estruturas (presente em MILC)

```
typedef struct {int re,imag} complexo;
complexo num1, num2, num3;
num3.re = num1.re + num2.re ;
num3.imag = num1.imag + num2.imag ;
```

Operações com partes reais têm *stride* maior que 1

Benchmark de Vetorização (cont.)

- Fatores importantes para vetorização (cont.)

  a) Alinhamento dos dados em memória
    - Em alguns casos, pode ser útil alinhar os vetores em fronteiras de 16 bytes (128 bits = tamanho do registro vetorial)
    - Transformações possíveis para favorecer alinhamento
      - *Padding*: adição de posições extras
      - *Loop Peeling*: separação de algumas iterações iniciais ou finais, para que as demais iterações acessem dados alinhados
    - Alguns compiladores podem fazer isso automaticamente, mas o programador pode exigir, por garantia

Benchmark de Vetorização (cont.)

- Fatores importantes para vetorização (cont.)
  - b) Dependências entre iterações:
    - Em muitos casos, loop pode ser modificado para remover a dependência, através de transformações
      - loop distribution
      - loop interchange
      - etc, etc

    Ref: D. A. Padua and M .J. Wolfe, *Advanced Compiler Optimizations for Supercomputers*, Communications of ACM, 29(12), Dec.1986, pp.1184-1200

Benchmark de Vetorização (cont.)

- Fatores importantes para vetorização (cont.)

b) Dependências entre iterações:

- Outras vezes, podem ser removidas mudando o algoritmo

Exemplo: `for (i=1; i<N; i++) a[i]=a[i-1]+a[i];` (recorrência)

Suponha valores iniciais a={10,11,12,13,14,...}:

`i=1: a[1]=a[0]+a[1]` {10,21,12,13,14,...}

`i=2: a[2]=a[1]+a[2]` {10,21,33,13,14,...}

`i=3: a[3]=a[2]+a[3]` {10,21,33,46,14,...}

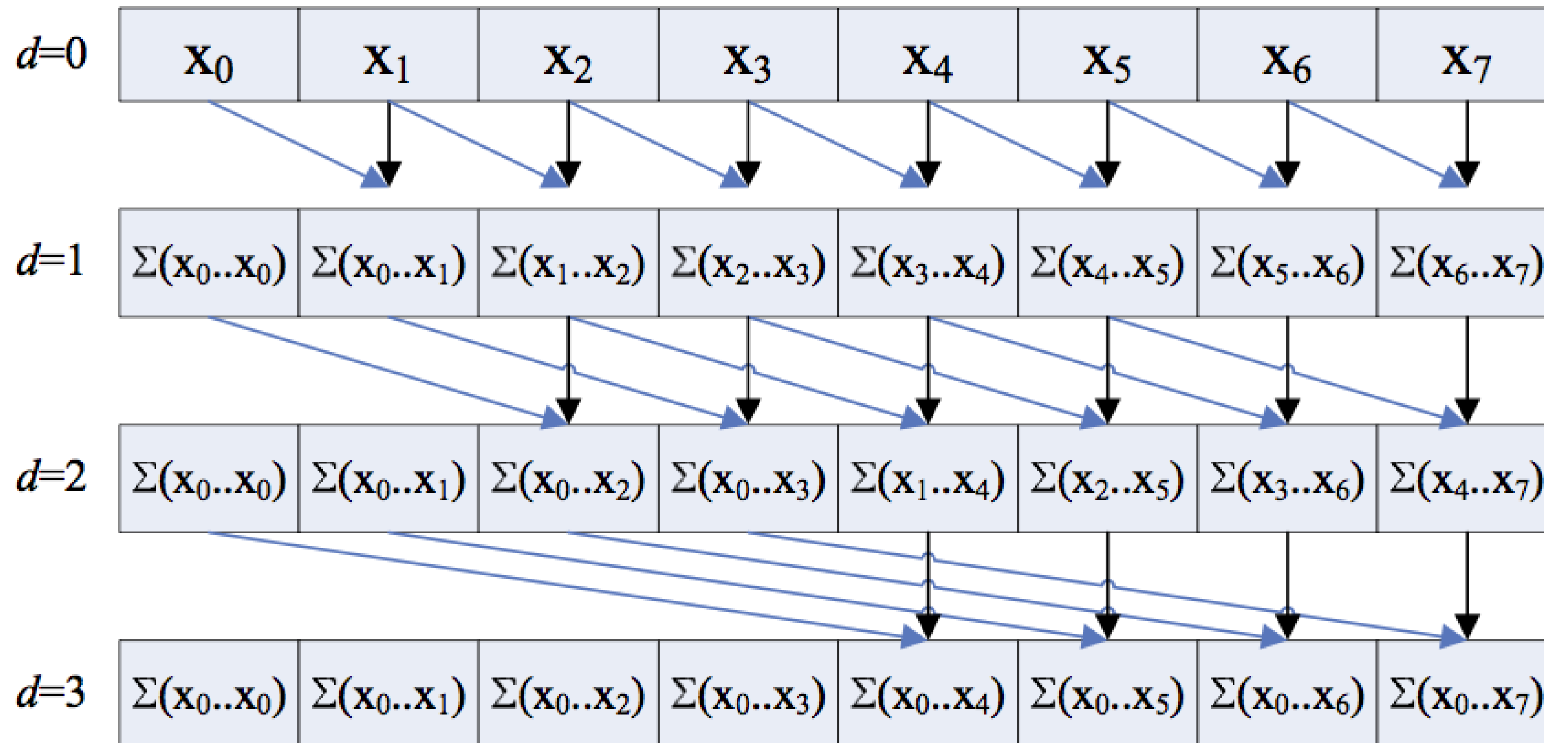
`...`

Ao final, cada elemento a[i] conterá a soma a[0]+a[1]+...+a[i-1]+a[i]

Pergunta: Como acelerar isso? (ou seja, rodar em menos que N-1 passos?)

Benchmark de Vetorização (cont.)

- Fatores importantes para vetorização (cont.)
  - Soma com prefix (*prefix sum*)



Benchmark de Vetorização (cont.)

- Fatores importantes para vetorização (cont.)
  - Soma com prefix (*prefix sum*)
    - Permite executar a mesma operação em log(N) passos
    - Ex: N=1 milhão → log(N)=20 passos !
    - Implementação Vetorial:
      - Passo 1: vetor $\{x_0,x_1,x_2,\ldots x_{N-1}\}$ somado a vetor $\{x_1,x_2,x_3,x_4,\ldots,x_N\}$
      - Passo 2: vetor $\{x_0,x_1,x_2,\ldots,x_{N-2}\}$ somado a vetor $\{x_2,x_3,x_4,x_5,\ldots,x_N\}$
      - Passo 3: vetor $\{x_0,x_1,x_2,\ldots,x_{N-4}\}$ somado a vetor $\{x_4,x_5,x_6,\ldots,x_N\}$
      - A cada passo i, $2^{(i-1)}$ elementos iniciais não são mais atualizados
      - Primeiro passo: some de vetores de comprimento N-1
      - Último passo: soma de vetores de comprimento N/2

Benchmark de Vetorização (cont.)

- TSVC: Maleki et al – plataformas e chaves de compilação

Platform	Model	Frequency (MHz)	Vector Unit	Vector Length	L3 Cache (MB)	Operating System
IBM	Power7 9179-MHB	3864.00	AltiVec	128 (bit)	32 (4 local)	Linux v2.6
Intel	Intel Core i7 920	2659.964	SSE4.2	128 (bit)	8	Linux v2.6

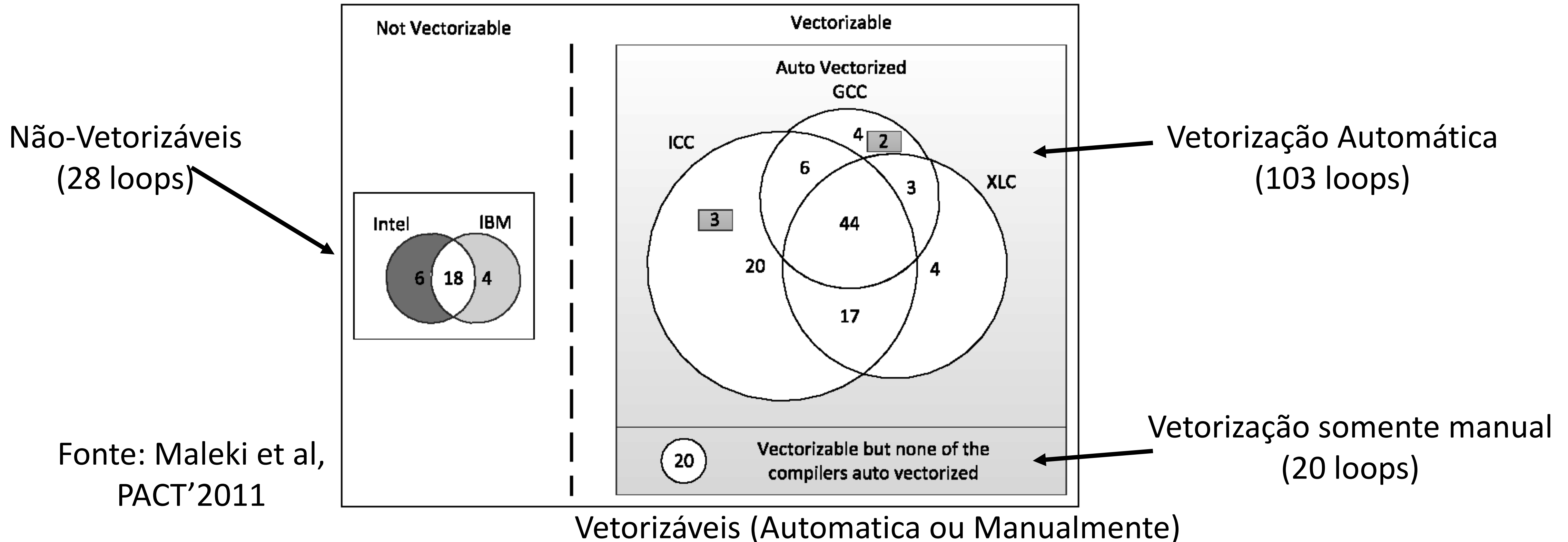
TABLE I
SPECIFICATION OF EXPERIMENTAL MACHINES

Specification	GCC	ICC	XLC
Version	4.7.0	12.0	11.1
Baseline Optimization	`-O3 -fivopts -funsafe-math-optimizations`	`-O3`	`-O3 -qhot -qarch=pwr7 -qtune=pwr7 -qipa=malloc16`
Vectorization Options	`-flax-vector-conversions -msse4.2`	`-xSSE4.2`	`-qenablevmx -qdebug=NSIMDCOST -qdebug=alwaysspec`
Disable Vectorization	`-fno-tree-vectorize`	`-no-vec`	`-qnoenablevmx`
Vectorization Report	`-ftree-vectorizer-verbose=[n]`	`-vec_report[n]`	`-qreport`

TABLE III
COMPILERS SPECIFICATIONS

Benchmark de Vetorização (cont.)

- Resultados com os 151 loops de TSVC:



Benchmark de Vetorização (cont.)

- Formas de vetorização, por compilador:

Method	XLC	ICC
Vectorizable	124(82.12%)	127(84.11%)
Perfectly Auto Vectorized	66(43.71%)	82(54.31%)
Source Level Transformation	42(27.82%)	38(25.17%)
Intrinsics	16(10.6%)	7(4.64%)

TABLE IV
LOOP CLASSIFICATION BASED ON THE METHOD USED TO ACHIEVE THE BEST SPEEDUP.

- XLC: 124 = 151 – (18+4+3+2)
- ICC: 127 = 151 – (6+18)
- Apenas 43.7% (XLC) ou 54.3% (ICC) são automáticos!

Benchmark de Vetorização (cont.)

- Ganhos de desempenho com vetorização:
  - Média calculada sobre os 151 loops
    - Loops não-vetorizados: S=1.0

Method	XLC	ICC	GCC
Auto Vectorization	1.66	1.84	1.58
Transformations	2.97	2.38	
Intrinsics	3.15	2.45	

TABLE VIII
THE AVERAGE SPEEDUPS OBTAINED BY EACH METHOD.

Testes com Aplicações Reais

- Ganhos de desempenho com aplicações reais

Method	XLC	ICC	GCC
Auto Vectorization	1.154	1.279	1.232
Manual	2.101	2.743	2.692

TABLE IX
THE AVERAGE SPEEDUP OBTAINED WITH AUTO VECTORIZATION AND MANUAL VECTORIZATION FOR PACT AND MEDIA BENCH II.

- Total de loops para todas as aplicações: 33
- Sucessos na vetorização automática pelo compilador:
  - XLC: 6 loops (18,2%)
  - ICC: 10 loops (30,3%)
  - GCC: 7 loops (21,2%)
  - Coletivamente (3 compiladores): 16 loops (48,49%)

Compiladores Vetorizadores

- Resumo:
  - Vetorização via compiladores é indispensável para obter portabilidade
  - Compilador sozinho pode ser insuficiente
    - Mudança no código pelo programador pode ser necessária para atingir maiores ganhos de desempenho
  - Diferentes compiladores têm capacidades distintas
    - Cada compilador otimiza um conjunto limitado de aspectos
    - Compiladores continuam evoluindo na sua capacidade de vetorização – cenário similar ao de 3 décadas atrás!