Curso: MI / MEI **Prova de Avaliação** 14/Dez/10
Disciplina: CPD, 1º semestre **Duração:** 2h30m

1. O modelo de computação proposto por von Neumann nos anos 40 assentava num conjunto de princípios, que incluíam as noções de "*instruction set*", de "*stored program*" e consequente "*program counter*", e de "memoria de programa e de dados". Von Neumann reconheceu ainda a necessidade de paralelismo nos computadores, mas foi pragmático ao reconhecer as dificuldades na sua concretização, optando por uma implementação sequencial. Este modelo manteve-se actual por várias décadas.
Os avanços da miniaturização da electrónica – com os circuitos integrados a conterem cada vez mais transístores – permitiram finalmente nos anos 70 e 80 a introdução de formas implícitas de paralelismo na arquitectura de um processador, com destaque para o bloco responsável pela execução das instruções em binário.

a)

A evolução ao nível da electrónica não foi igual para as 2 principais componentes de um computador: o CPU e a memória. Há um *gap* crescente entre o desempenho destes 2 elementos, ao nível da latência nos acessos à memoria, quando medida em termos de períodos de *clock*.

b)

2. Considere o cálculo do factorial de um inteiro, e a execução do código associado numa arquitectura com as características da apresentada no problema anterior. Vários algoritmos podem ser usados na construção do código, sendo os mais comuns os que usam uma função recursiva e os que usam um método iterativo, tal como o que é apresentado aqui:

```
int fact(int n)
{
   int i;
   int result = 1;
   for (i = n; i > 0; i--)
      result = result * i;
   return result;
}
```

Com *loop unrolling* este mesmo código pode ser reescrito assim:

```
int fact_u2(int n)
{
   int i;
   int result = 1;
   for (i = n; i > 1; i-=2)
      result = (result * i) * (i-1);
   return result;
}
```

ou assim (apenas alterou o interior do ciclo):

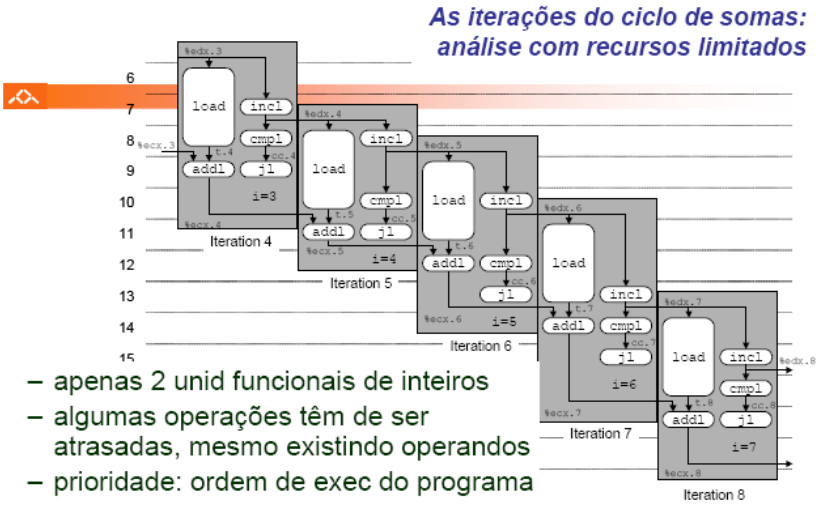
```
int fact_u2a(int n)
{
   int i;
   int result = 1;
   for (i = n; i > 1; i-=2)
      result = result * (i * (i-1));
   return result;
}
```

a)

b)

c)

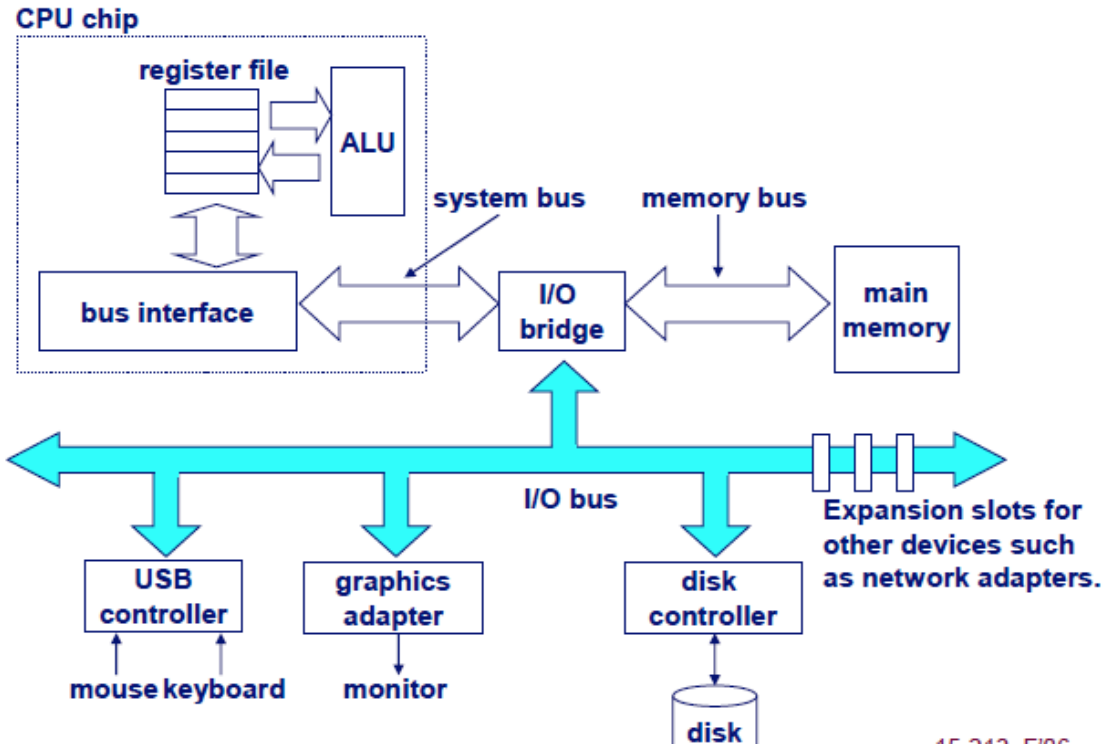
Ilustre a justificação para melhor clareza de leitura, de forma semelhante à usada na bibliografia (em baixo).



3. Considere o estudo feito para a análise comparativa dos processadores mais recentes da Intel e da AMD da linha x86.

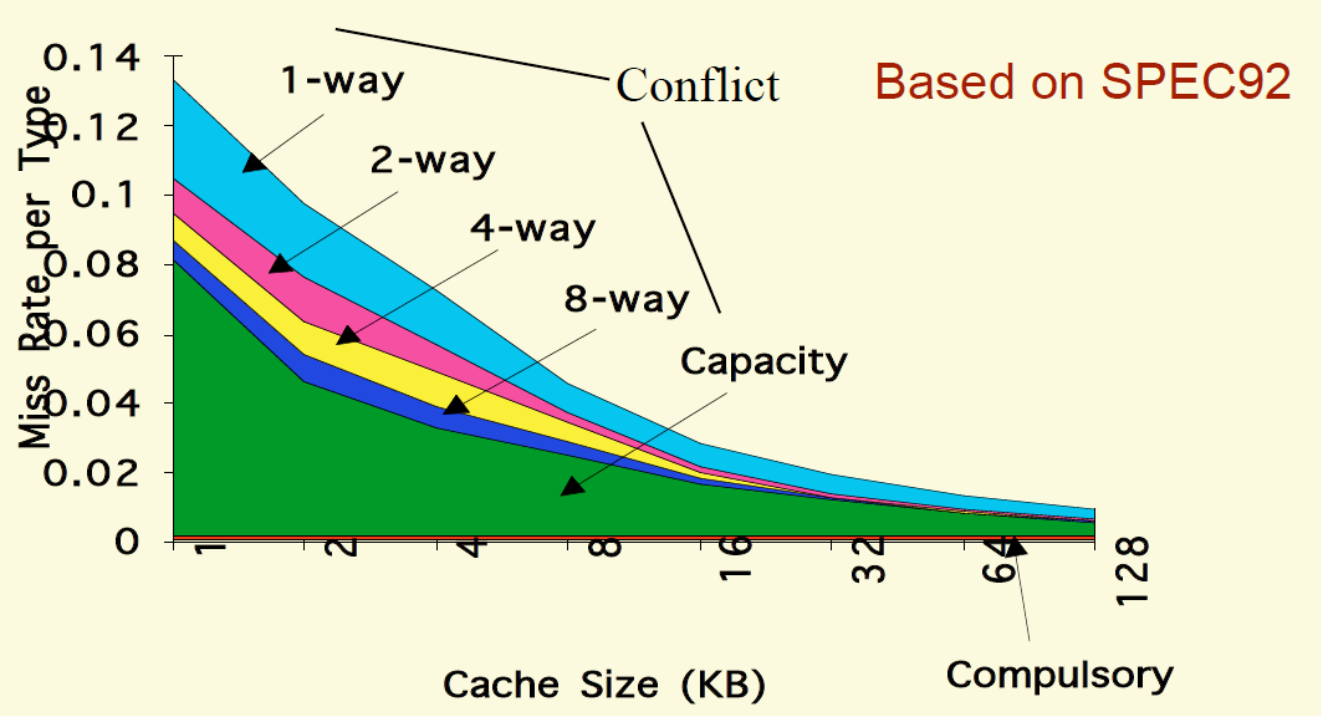
a)

Considere ainda a seguinte figura:



b)

Considere agora figura seguinte:



c)

4. O ambiente CUDA pressupõe um modelo de arquitectura subjacente e disponibiliza extensões a várias linguagens de programação de CPUs. O código de multiplicação de matrizes em baixo foi redigido para execução em ambiente CUDA.

```
#define TILE_WIDTH = 16; // Dimensão do bloco

__global__ void cudaMMV1(float* Ad, float* Bd, float* Cd, int Width) {

  __shared__float Ads[TILE_WIDTH][TILE_WIDTH];
  __shared__float Bds[TILE_WIDTH][TILE_WIDTH];
  int bx = blockIdx.x; int by = blockIdx.y;
  int tx = threadIdx.x; int ty = threadIdx.y;
  int Row = by * TILE_WIDTH + ty;
  int Col = bx * TILE_WIDTH + tx;
  float Pvalue = 0;
  for (int m = 0; m < Width/TILE_WIDTH; ++m) {
       Ads[ty][tx] = Ad[Row][m*TILE_WIDTH + tx];
       Bds[ty][tx] = Bd[m*TILE_WIDTH + ty][Col];

       __Syncthreads();

       for (int k = 0; k < TILE_WIDTH; ++k)
            Pvalue += Ads[ty][k] * Bds[k][tx];
       Cd[Row][Col] = Pvalue;
  }
}

__global__ void cudaMMV2(float* Ad, float* Bd, float* Cd, int Width) {

  __shared__float Ads[TILE_WIDTH][TILE_WIDTH];
  __shared__float Bds[TILE_WIDTH][TILE_WIDTH];
  int bx = blockIdx.x; int by = blockIdx.y;
  int tx = threadIdx.x; int ty = threadIdx.y;
  int Row = by * TILE_WIDTH + ty;
```

```
    int Col = bx * TILE_WIDTH + tx;
    float Pvalue = 0;
    for (int m = 0; m < Width/TILE_WIDTH; ++m) {
         Ads[ty][tx] = Ad[Row][m*TILE_WIDTH + tx];
         Bds[ty][tx] = Bd[m*TILE_WIDTH + ty][Col];

         __Syncthreads();

         Pd[Row][Col] = Ads[ty][0] * Bds[0][tx]
                + Ads[ty][1] * Bds[1][tx]
                + Ads[ty][2] * Bds[2][tx]
                + (…) +
              + Ads[ty][14] * Bds[14][tx]
              + Ads[ty][15] * Bds[15][tx];
    }
}
```

a)

b)

c)

Folha de resolução

Nome: Folha de

Folha de resolução

Nome: Folha de

Folha de resolução

Nome: Folha de