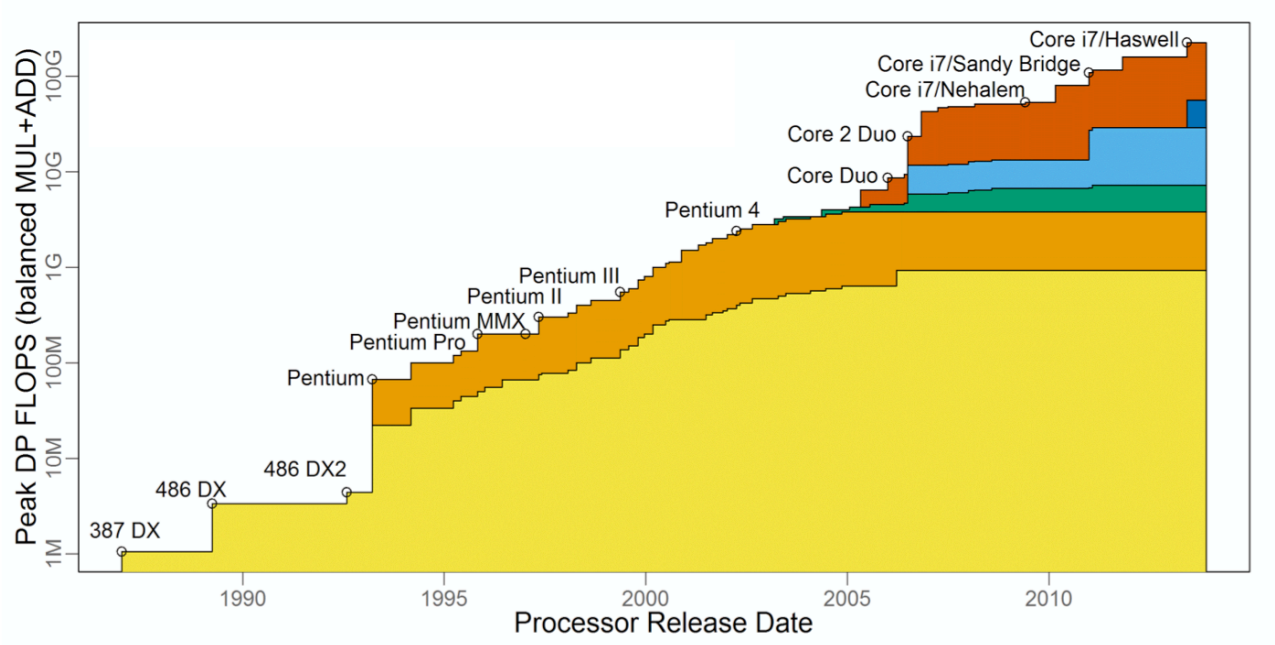
Curso: MIEI / MEI
U.C.: CPD/AA, 1º semestre

Exame 22/jan/19
Duração: 2h30m

1. [80] **Comente/explique** as seguintes afirmações, incluindo implicações e consequências:
   - **a)** [15] "*The L3 cache in Xeon devices is also organized as n-way associative as the L2 cache and the only difference between them is the size of the cache*".
   - **b)** [15] *"To get the best performance of the dot product of 2 arrays, declared as global variables and passed as arguments (pointers) to a C function, all that is needed is to compile this code with* `-O3`".
   - **c)** [15] *"The only relevant feature that was improved when Intel moved from AVX2 to AVX-512 was the size of the vector operands, from 256 bits to 512 bits"*.
   - **d)** [15] "*The max number of warps/multiprocessor ready to be executed in the scoreboard went from 48 to 64 when the GPU NVidia Fermi was updated to the Kepler architecture*".
   - **e)** [20] "*A Intel conseguiu passar da arquitetura Xeon com menos de 30 cores para a arquitetura Xeon Phi com mais de 60 cores e 2 VPU/core reduzindo o tamanho dos transístores, usando DRAM nas caches e tecnologia 3D na construção do chip*".

2. [20] Considere a arquitetura do *package* da Intel com o *chip* KNL (com 64 *cores*, a 1GHz, e cada com 2 unidades vetoriais de 512 bits) e com 8 *chips* de memória embebida 3D.
   Considere o início da execução de um pedaço de código em cada um dos cores (com *cache* fria), contendo uma adição de *floating-point* sobre vetores de precisão dupla, uniformemente distribuídos pelos 8 *chips* de memória embebida, e com os seus elementos em posições contíguas de memória.
   Calcule a largura de banda que cada um dos controladores de memória deveria ser capaz de fornecer os elementos do vetor para que cada *core* espere o mínimo tempo possível para aceder aos dados.

3. [20] Considere a seguinte figura que mostra a evolução dos processadores x86 da Intel, em termos de desempenho na execução de instruções de FP precisão dupla:



Identifique e caracterize sumariamente os principais 5 factores que contribuíram para este aumento do desempenho.

4. [45] Considere o seguinte código de multiplicação de matrizes, com otimização por blocos:

```
1 void regularMatrixMult (void) {
2     for (unsigned i = 0; i < SIZE; ++i) {
3         for (unsigned j = 0; j < SIZE; ++j) {
4               result[i][j] = 0;
5               for (unsigned k = 0; k < SIZE; ++k) {
6                     result[i][j] += m1[i][k] * m2[k][j];
7               }
8         }
9     }
10 }
```

Quando compilado com o comando `icc -vec-report=5` produziu a seguinte lista de comentários:

```
LOOP BEGIN at line 2 inlined into main
<Distributed into 2 chunks>
   remark #25444: Loopnest Interchanged: ( 1 2 3 ) --> ( 1 3 2 )
   (...)
<Remainder loop for vectorization>
   remark #15388: vectorization support: reference result[i][j] has aligned access
   (...)
   remark #15381: vectorization support: unaligned access used inside loop body
   remark #15305: vectorization support: vector length 16
   remark #15309: vectorization support: normalized vectorization overhead 0.357
   remark #15301: REMAINDER LOOP WAS VECTORIZED
LOOP END
```

a) [15] **Apresente uma explicação** para o compilador ter trocado a ordem de execução dos ciclos da função.

b) [15] As *miss rates* das multiplicações de matrizes no modo **n**ormal e com otimização por **b**locos são de $L2_n$ = 5%, $L3_n$ = 20% e $L2_b$ = 95%, $L3_b$ = 7%. **Indique**, justificando, as razões para a diminuição da *miss rate* na *cache* L2 e o aumento da *miss rate* na L3 na versão por blocos.

c) [15] **Indique**, justificando, as melhorias de *performance* que a vectorização tem numa multiplicação de matrizes por blocos, referindo possíveis limitações.

5. [35] O alinhamento e estruturação dos dados na memória tem um papel fundamental para a execução eficiente de código.

a) [20] **Desenvolva** uma implementação simples de um algoritmo de *stencil* em CUDA. Para cada elemento do *array* de entrada `in`, o *kernel* deve somar o seu valor aos 4 elementos vizinhos (2 elementos à sua esquerda e 2 à direita) e colocar o resultado em `out`.

```
__global__
void stencil (float *in, float *out, float vector_size) {
  unsigned tid = treadId.x + blockId.x * blockDim.x;
  // ...
}
```

b) [15] **Caracterize** as diferenças entre o alinhamento de dados em CPU, essenciais para a vectorização, e os acessos à memória *"coalesced"* em GPU.

Folha de resolução

Nome: Folha de

Folha de resolução

Nome: Folha de

Folha de resolução

Nome: Folha de