Aula 03

STL , Vector e Tilling

Prof. Lícia Sales Costa Lima
V 1.0

Recapitulando...

Arquitetura Cluster Franky

SRUN

srun --nodelist=compute10 --partition=normal --ntasks=1 --pty bash

```
who-am-I@compute10:$ Terminal dentro do nó de computação
who-am-I@compute10:$ ls
 todos     os.py     seus.txt    arquivos.bin        e_binários.cpp      sao_acessíveis.sh
Aqui.sbatch          s2.txt                  out.out
```

srun --partition=normal --ntasks=4 ./meu_programa_paralelo

```
############################################################
PRINTS DO SEU CÓDIGO SERÃO EXIBIDOS NO TERMINAL

Seu código será executado em um nó de computação e você poderar interagir com
seus outputs pelo terminal.
```

Insper

SLURM - SBATCH

Enquanto trabalhamos com código sequenciais, nosso arquivo .*slurm* deve ser:

```
#!/bin/bash
#SBATCH --job-name=nome_que_vc_quiser
#SBATCH --output=nome_do_aquivo_de_saida_%j.txt
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=quantidade_de_memoria_RAM
#SBATCH --time=tempo_do_seu_codigo
#SBATCH --partition=nome_da_fila
```

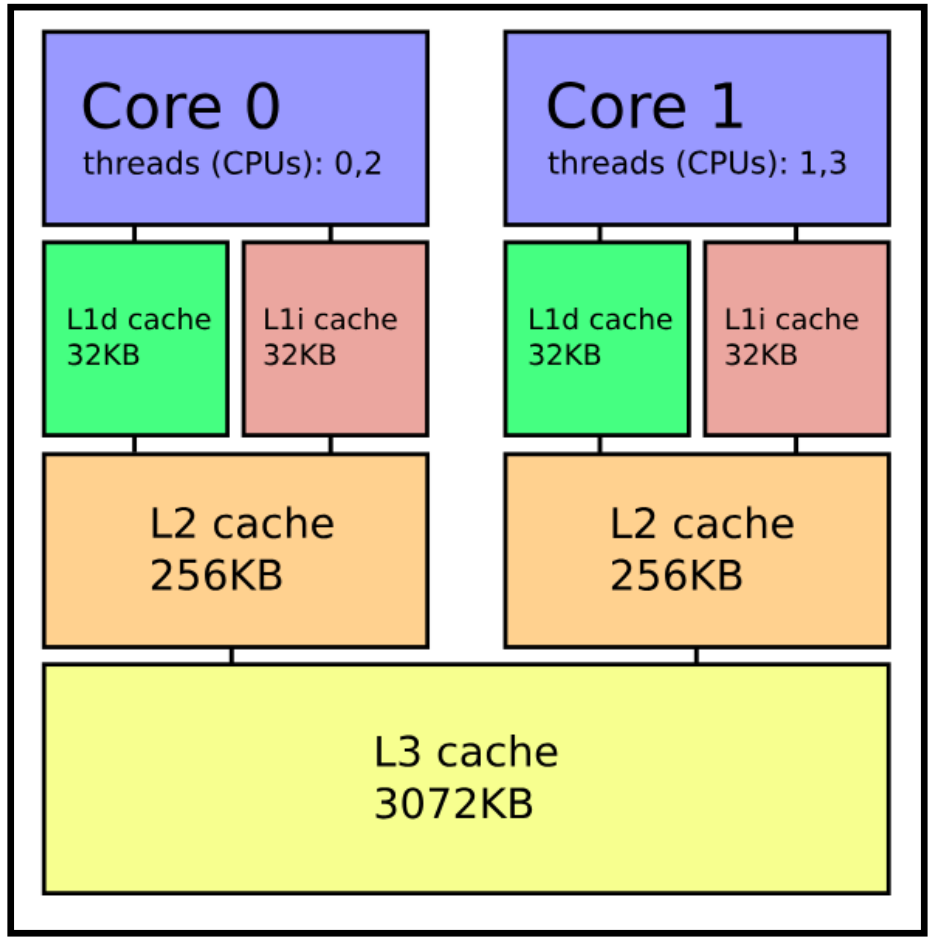
Insper

Mapa de Memória

Insper

CPU ↔ L1 ↔ L2 ↔ L3 ↔ RAM.

- L1: muito rápida, mas pequena.
- L2: intermediária.
- L3: maior, mas mais lenta.
- RAM: muito maior, mas centenas de vezes mais lenta.



Principio da Localidade Temporal

Insper

Principio da Localidade Temporal

Se uma variável foi acessada recentemente, há grandes chances dela ser acessada de novo em breve.

Quando você roda um loop que soma várias vezes a mesma variável,
essa variável fica na cache e é usada repetidamente sem precisar buscar de novo na RAM.

```c
for (int i = 0; i < 1000; i++) {
    sum += array[i];
}
```

Nesse loop, a variável **sum** é acessada repetidamente em todas as operações

Insper

Princípio da Localidade Espacial

Insper

Principio da Localidade Espacial

Se uma variável foi acessada, é provável que outros dados próximos na memória também sejam acessados logo em seguida.

Ao percorrer um vetor em ordem, a CPU já traz um “bloco” de elementos para a cache, aproveitando que você vai usar os vizinhos.

```c
for (int i = 0; i < 1000; i++) {
    sum += array[i];
}
```

Quando a CPU acessa **array[0]**, ela não traz apenas esse elemento, ele traz um bloco de dados vizinhos

Sem considerar Principio da Localidade Espacial

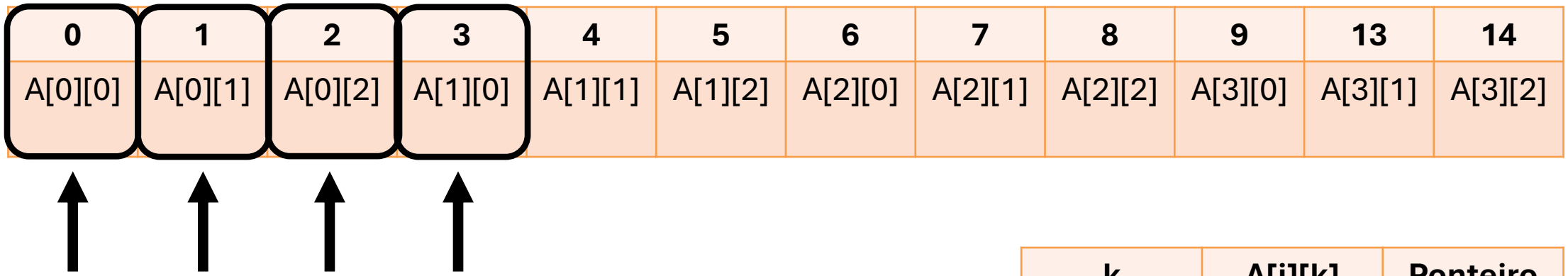
- **A[i][k]** é acessado em ordem contígua (boa localidade espacial).
- Mas **B[k][j]** é acessado pulando dados (péssima localidade espacial.)

k	A[i][k]	Ponteiro
0	A[0][0]	0
1	A[0][1]	1
2	A[0][2]	2
3	A[0][3]	3
...	...	...

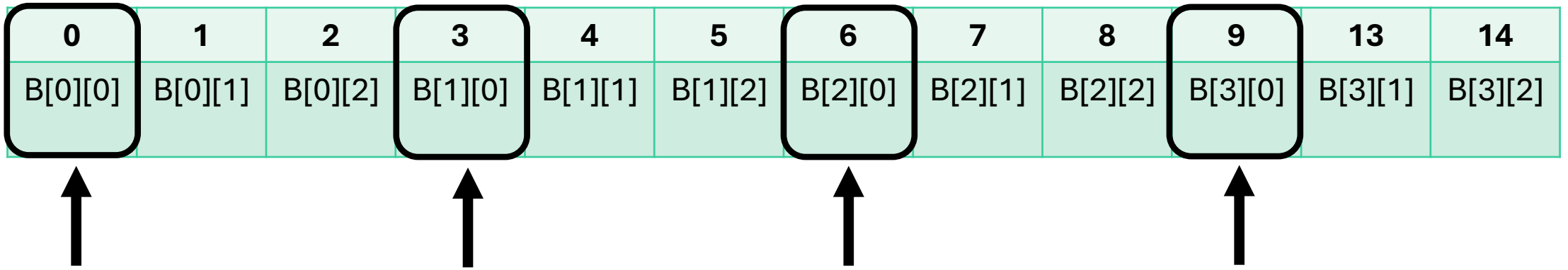
```cpp
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        for (int k = 0; k < N; k++) {
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

k	B[k][j]	Ponteiro
0	B[0][0]	0
1	B[1][0]	3
2	B[2][0]	6
3	B[3][0]	9
...	...	...

Insper



k	A[i][k]	Ponteiro
0	A[0][0]	0
1	A[0][1]	1
2	A[0][2]	2
3	A[0][3]	3
...	...	...



k	B[k][j]	Ponteiro
0	B[0][0]	0
1	B[1][0]	3
2	B[2][0]	6
3	B[3][0]	9
...	...	...

Considerando Principio da Localidade Espacial

```cpp
for (int i = 0; i < N; i++)
  for (int k = 0; k < N; k++) {
    double aik = A[i][k];              // reuso (temporal)
    for (int j = 0; j < N; j++)
      C[i][j] += aik * B[k][j];        // B por linha (contíguo)
  }
```

k	A[i][k]	Ponteiro
0	A[0][0]	0
1	A[0][1]	1
2	A[0][2]	2
3	A[0][3]	3
...	...	...

k	B[k][j]	Ponteiro
0	B[0][0]	0
1	B[0][1]	1
2	B[0][2]	2
3	B[0][3]	3
...	...	...

- **A[i][k] e B[k][j]** é acessado em ordem contígua

0	1	2	3	4	5	6	7	8	9	13	14
A[0][0]	A[0][1]	A[0][2]	A[1][0]	A[1][1]	A[1][2]	A[2][0]	A[2][1]	A[2][2]	A[3][0]	A[3][1]	A[3][2]

0	1	2	3	4	5	6	7	8	9	13	14
B[0][0]	B[0][1]	A[0][2]	B[1][0]	B[1][1]	B[1][2]	B[2][0]	B[2][1]	B[2][2]	B[3][0]	B[3][1]	B[3][2]

k	A[i][k]	Ponteiro
0	A[0][0]	0
1	A[0][1]	1
2	A[0][2]	2
3	A[0][3]	3
...	...	...

k	B[k][j]	Ponteiro
0	B[0][0]	0
1	B[0][1]	1
2	B[0][2]	2
3	B[0][3]	3
...	...	...

Tiling – Fatiamento de dados

- É a técnica de quebrar a matriz em blocos menores (submatrizes).
- Cada tamanho de bloco é escolhido para caber na cache, evitando que a CPU precise buscar dados da RAM o tempo todo.

Insper

Afinal, como saber quanto cabe na memória?

Get Memory Size

You can check how much memory a variable type uses with the `sizeof` operator:

Example

```
#include <iostream>
using namespace std;

int main() {
  int myInt;
  float myFloat;
  double myDouble;
  char myChar;

  cout << sizeof(myInt) << "\n";     // 4 bytes (typically)
  cout << sizeof(myFloat) << "\n";   // 4 bytes
  cout << sizeof(myDouble) << "\n";  // 8 bytes
  cout << sizeof(myChar) << "\n";    // 1 byte
  return 0;
}
```

Try it Yourself »

Fonte: https://www.w3schools.com/cpp/cpp_memory_management.asp

Tomando como base o hardware do monstrão, ele tem um processador **Intel Xeon Gold 5215**, que possui:
L1d cache: 32 KiB por núcleo
L2 cache: 1 MiB por núcleo
L3 cache: 13.75 MiB por socket

L1d = 32 KiB = 32 × 1024 = 32768 bytes
Cada double = 8 bytes

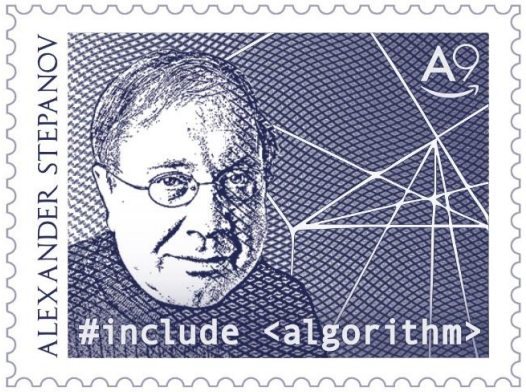
$$\text{Nº de doubles na L1} = \frac{32768}{8} = 4096$$

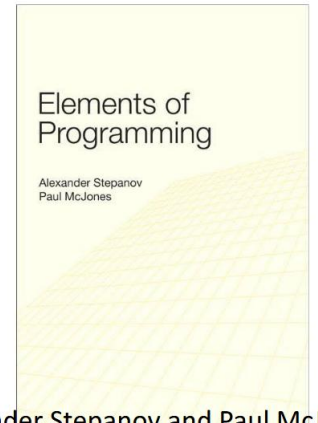
Ou seja, até 4096 doubles caberiam, se só a sua matriz estivesse ocupando a cache.

STL
The Standard Template Library

Insper

using namespace std





Alexander Stepanov and Paul McJones. 2009. *Elements of Programming* (1st ed.). Addison-Wesley Professional.

C++ library headers				
<algorithm>	<iomanip>	<list>	<ostream>	<streambuf>
<bitset>	<ios>	<locale>	<queue>	<string>
<complex>	<iosfwd>	<map>	<set>	<typeinfo>
<deque>	<iostream>	<memory>	<sstream>	<utility>
<exception>	<istream>	<new>	<stack>	<valarray>
<fstream>	<iterator>	<numeric>	<stdexcept>	<vector>
<functional>	<limits>			
Headers added in C++11				
<array>	<condition_variable>	<mutex>	<scoped_allocator>	<type_traits>
<atomic>	<forward_list>	<random>	<system_error>	<typeindex>
<chrono>	<future>	<ratio>	<thread>	<unordered_map>
<codecvt>	<initializer_list>	<regex>	<tuple>	<unordered_set>
Headers added in C++14				
<shared_mutex>				
Headers added in C++17				
<any>	<execution>	<memory_resource>	<string_view>	<variant>
<charconv>	<filesystem>	<optional>		

.
.
.

https://en.cppreference.com/w/cpp/standard_library.html

Functions

`operator==` `operator!=` (removed in C++20) `operator<` (removed in C++20) `operator<=` (removed in C++20) `operator>` (removed in C++20) `operator>=` (removed in C++20) `operator<=>` (C++20)	lexicographically compares the values of two `vectors` (function template)
`std::swap`(`std::vector`)	specializes the `std::swap` algorithm (function template)
`erase`(`std::vector`) `erase_if`(`std::vector`) (C++20)	erases all elements satisfying specific criteria (function template)

Range access

`begin` (C++11) `cbegin` (C++14)	returns an iterator to the beginning of a container or array (function template)
`end` (C++11) `cend` (C++14)	returns an iterator to the end of a container or array (function template)
`rbegin` `crbegin` (C++14)	returns a reverse iterator to the beginning of a container or array (function template)
`rend` `crend` (C++14)	returns a reverse end iterator for a container or array (function template)
`size` (C++17) `ssize` (C++20)	returns the size of a container or array (function template)
`empty` (C++17)	checks whether the container is empty (function template)
`data` (C++17)	obtains the pointer to the underlying array (function template)

Includes

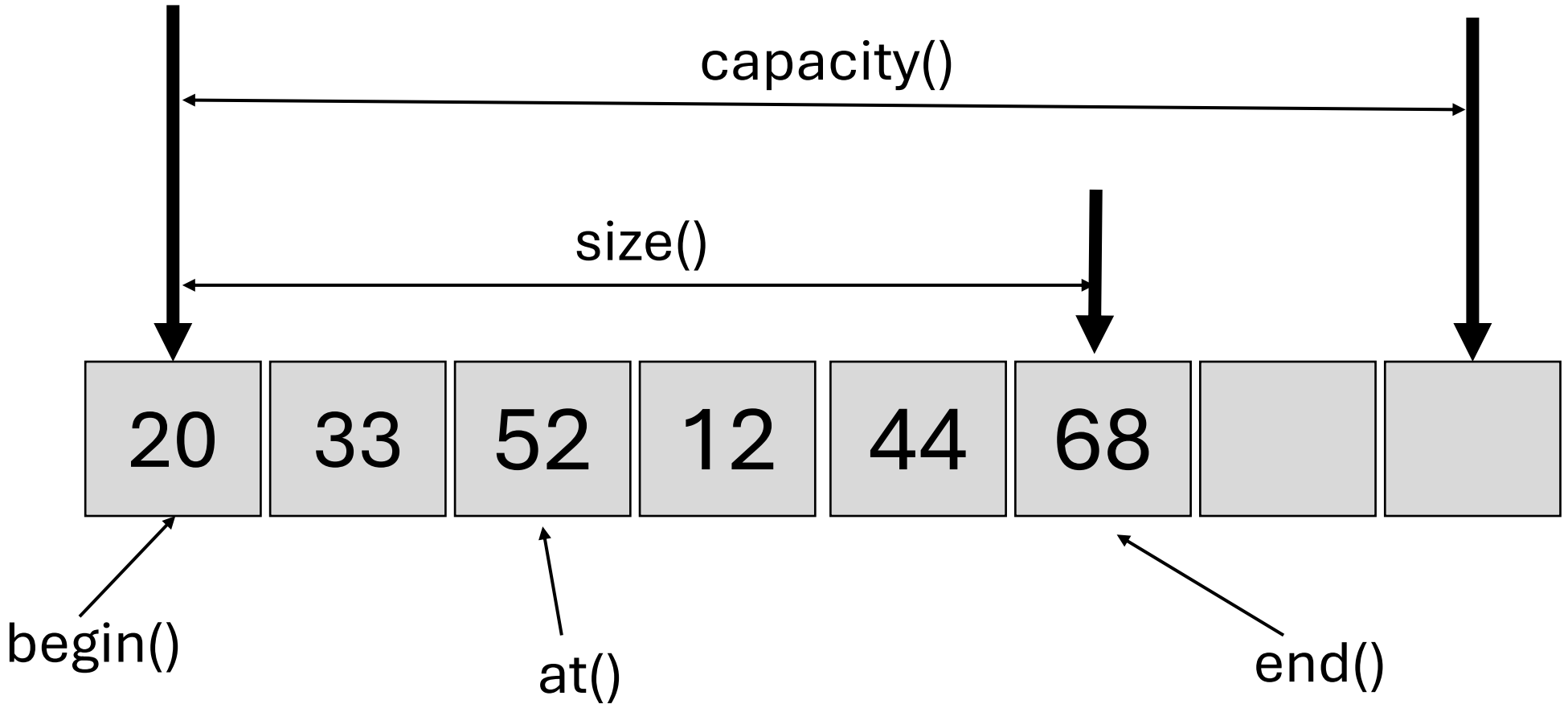
`<compare>` (C++20)	Three-way comparison operator support
`<initializer_list>` (C++11)	`std::initializer_list` class template

Classes

`vector`	resizable contiguous array (class template)
`vector<bool>`	space-efficient dynamic bitset (class template specialization)
`std::hash<std::vector<bool>>` (C++11)	hash support for `std::vector<bool>` (class template specialization)

https://en.cppreference.com/w/cpp/header/vector.html

```
std::vector<int> meu_vector = {20,33,52,12,44,68};
```




SUA VEZ