



Insper Supercomputação

Objetivos de aprendizagem

1. Criar implementações eficientes para problemas computacionalmente difíceis;
2. Planejar e projetar sistemas de computação de alto desempenho, escolhendo as tecnologias mais adequadas para cada tipo de aplicação;
3. Utilizar recursos de computação multi-core para melhorar o desempenho de programas sequenciais;
4. Implementar algoritmos ingenuamente paralelizáveis em GPU;
5. Analisar resultados de desempenho levando em conta complexidade computacional e tecnologias usadas na implementação.

Recursos computacionais

- GCC 8.0 (ou superior) -- C++11
- Linux (Ubuntu 18.04 ou superior)
- Monstrão (containers/VMs)
 - ambiente de testes padrão

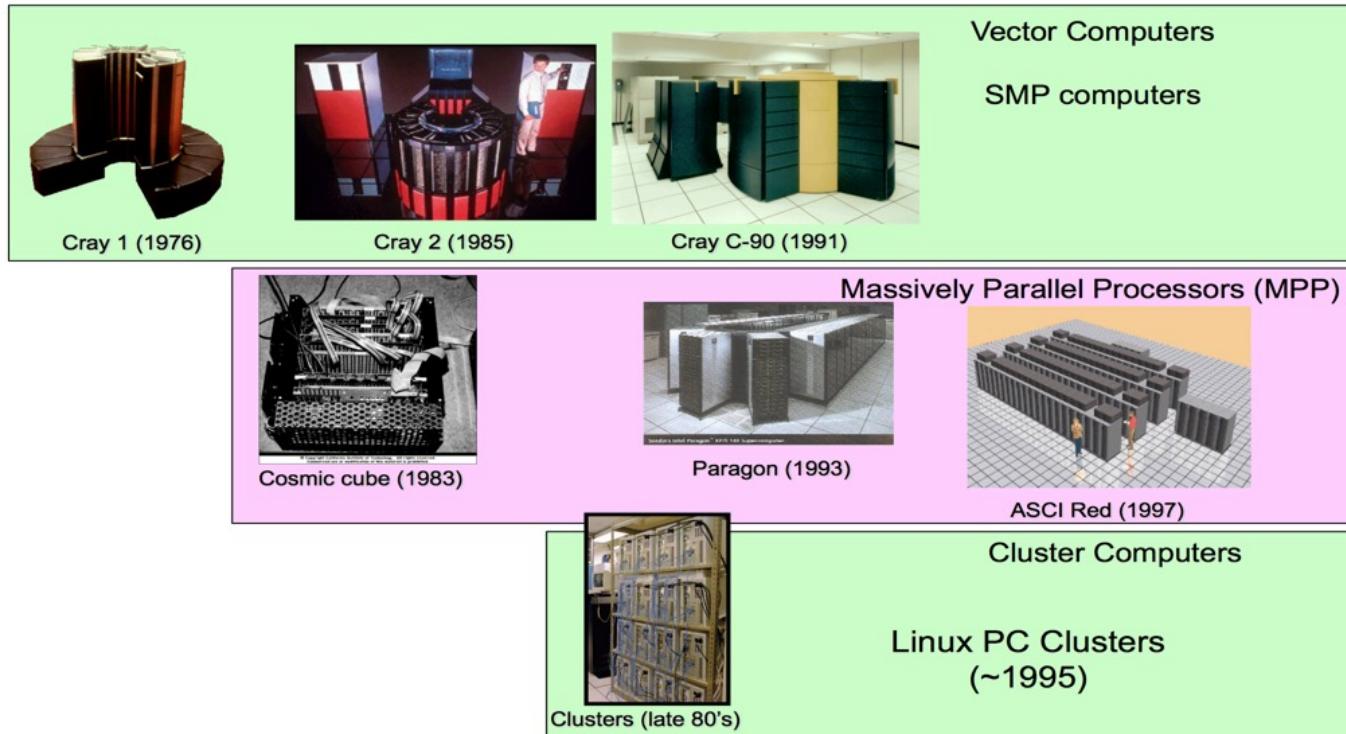
Problemática

- Algoritmos complexos são aplicados em diversas situações para orientar decisões de negócios e para otimizar a alocação/distribuição de recursos
- Um determinado algoritmo é atualmente considerado lento demais
- E agora?

O que é uma solução de alto desempenho?

1. Algoritmos eficientes
2. Implementação eficiente
 - o Cache, paralelismo de instrução
 - o Linguagem de programação adequada
3. Paralelismo

Paralelismo



Resolução no acesso a recursos

- Supercomputação sob demanda

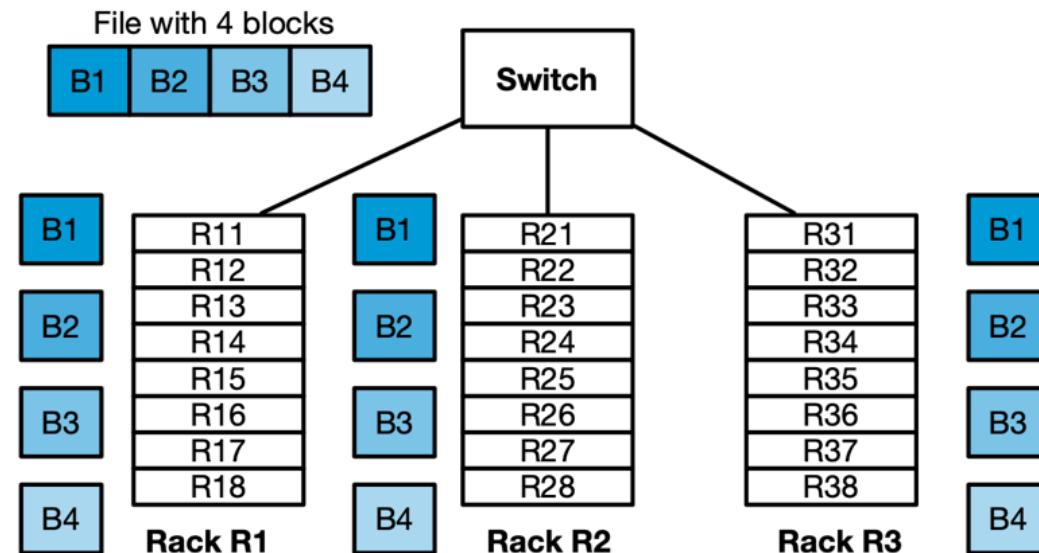


Escalabilidade

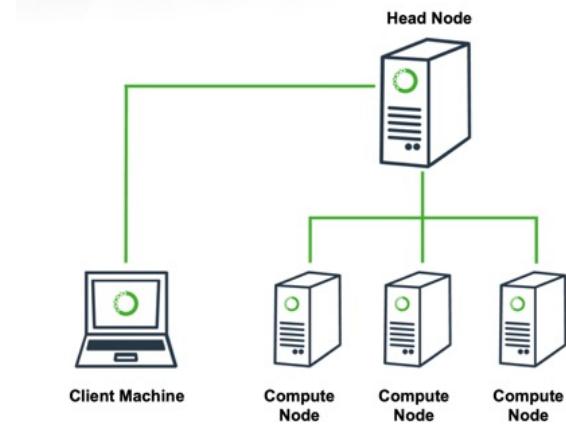
- É a habilidade de um sistema lidar com o aumento da carga de processamento sem apresentar uma degradação significante em seu desempenho
- Há duas formas de se obter a escalabilidade:
 - **Escalabilidade vertical – *scale-up***
 - Fazer upgrade na infra existente (+ memória, por exemplo)
 - **Escalabilidade horizontal – *scale-out***
 - Adicionar novas máquinas ao parque computacional
 - Distribuir os dados e o trabalho de processamento em diversas máquinas

Escalabilidade horizontal

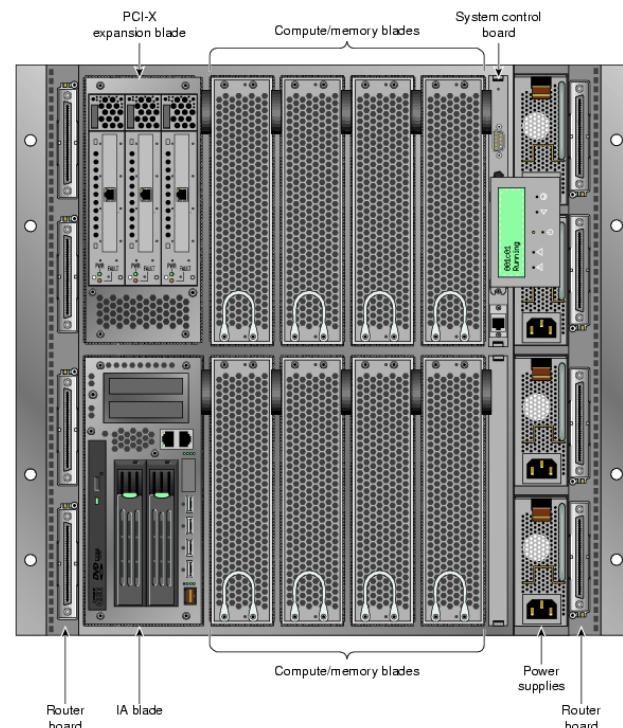
- Podemos fazer uso de um **cluster** de máquinas
- Os dados são armazenados em um sistema de arquivos distribuído (e.g., HDFS)
- Cada arquivo é dividido em blocos de tamanho fixos
- Cada bloco é **replicado** em diversos nós do cluster



Cluster



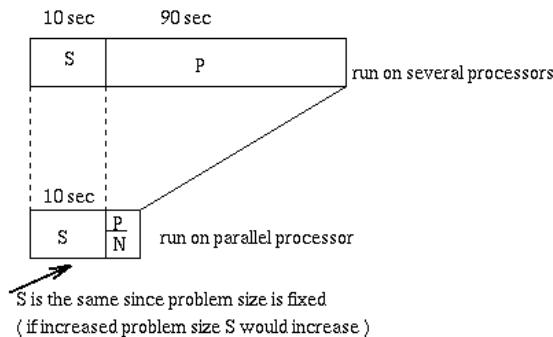
Supercomputador



Lei de Amdahl

- Numa aplicação existe sempre uma parte que não pode ser paralelizada
- Seja **S** a parte do trabalho sequêncial, **1-S** é a parte susceptível de ser paralelizada
- Mesmo que a parte paralela seja perfeitamente escalável, o aumento do desempenho (**speedup**) está limitado pela parte sequêncial

n = número de processadores



$$\text{Speedup} = \frac{1}{S + \frac{(1 - S)}{n}}$$

Speedup

- Se 10% das operações de um código precisam ser feitas sequencialmente, então o speedup não pode ser maior do que 10, independente do número de processadores

$$S = \frac{1}{0.1 + \frac{0.9}{10}} \approx 5.3$$

$p = 10$ processors

$$S = \frac{1}{0.1 + \frac{0.9}{\infty}} = 10$$

$p = \infty$ processors

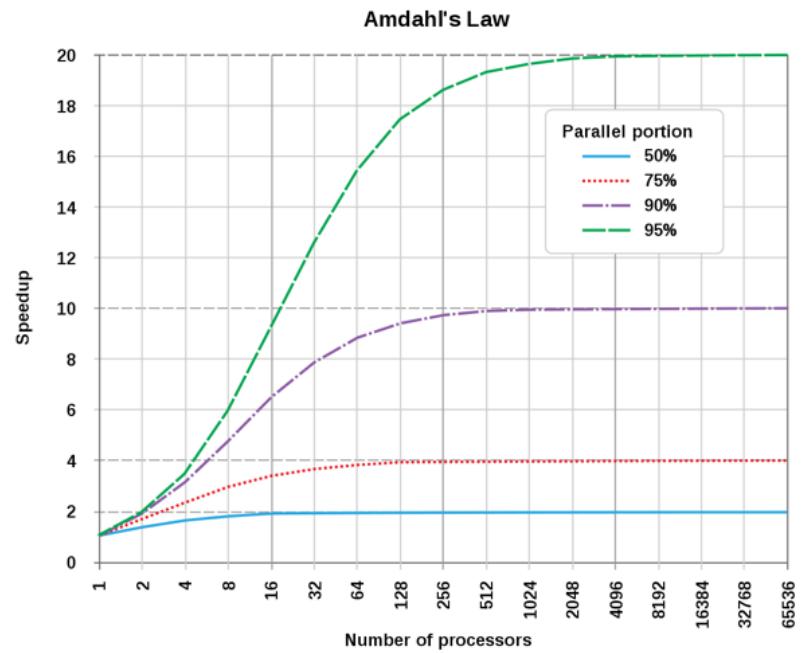


Imagen: Wikipedia

E o Big Data?

- Gartner:

*Big Data faz referência ao grande volume, variedade e velocidade de dados que **demandam formas inovadoras e rentáveis de processamento da informação**, para melhor percepção e tomada de decisão.*

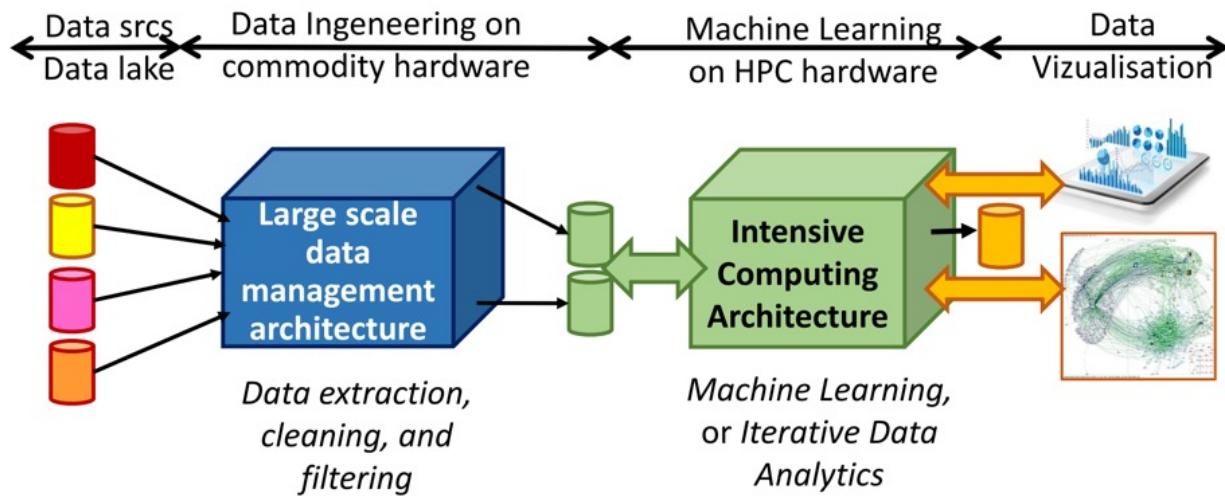
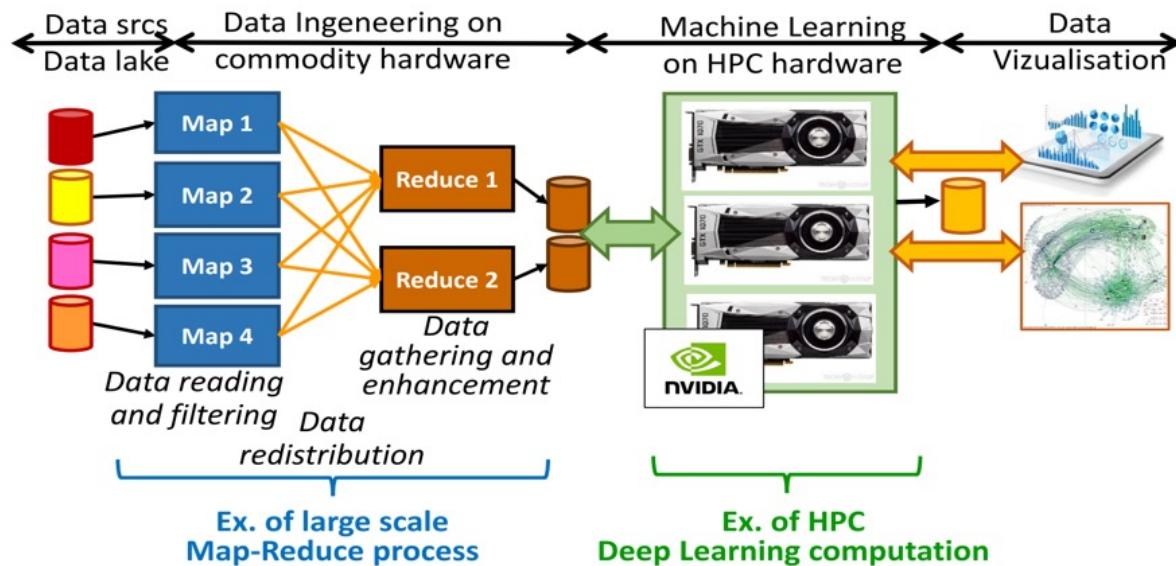


Imagen: Stéphane Vialle & Gianluca Quercini, CentraleSupélec – Université Paris-Saclay

E o Big Data?

- Gartner:

*Big Data faz referência ao grande volume, variedade e velocidade de dados que **demandam formas inovadoras e rentáveis de processamento da informação**, para melhor percepção e tomada de decisão.*





Atividade prática

- **Comparando soluções para um problema**

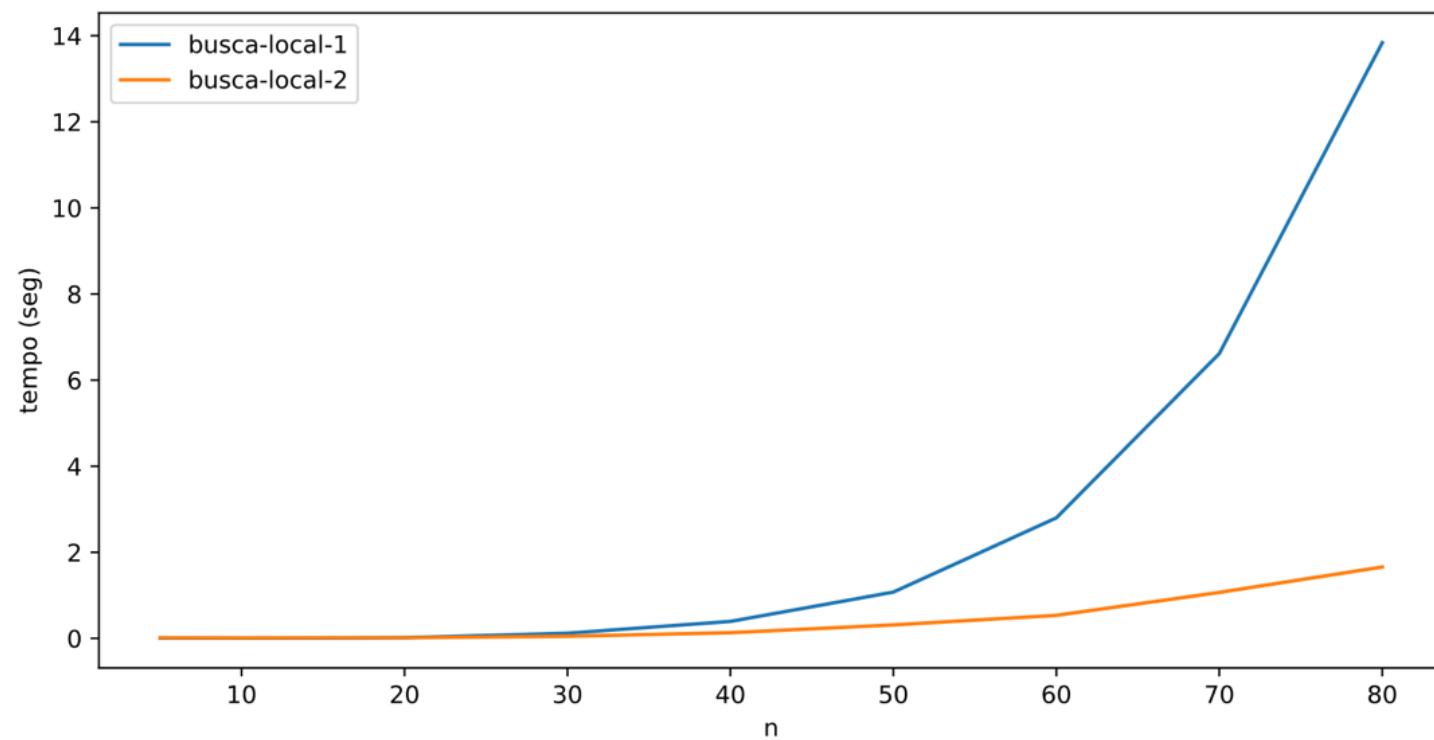
- Medir tempo de execução de programas usando Python
- Ordenar implementações de acordo com sua eficiência
- Discutir custo benefício de diferentes métodos de resolução de um problema.



Discussão

- **Discussão 1: o quanto um bom algoritmo faz diferença?**

Discussão – o algoritmo importa!

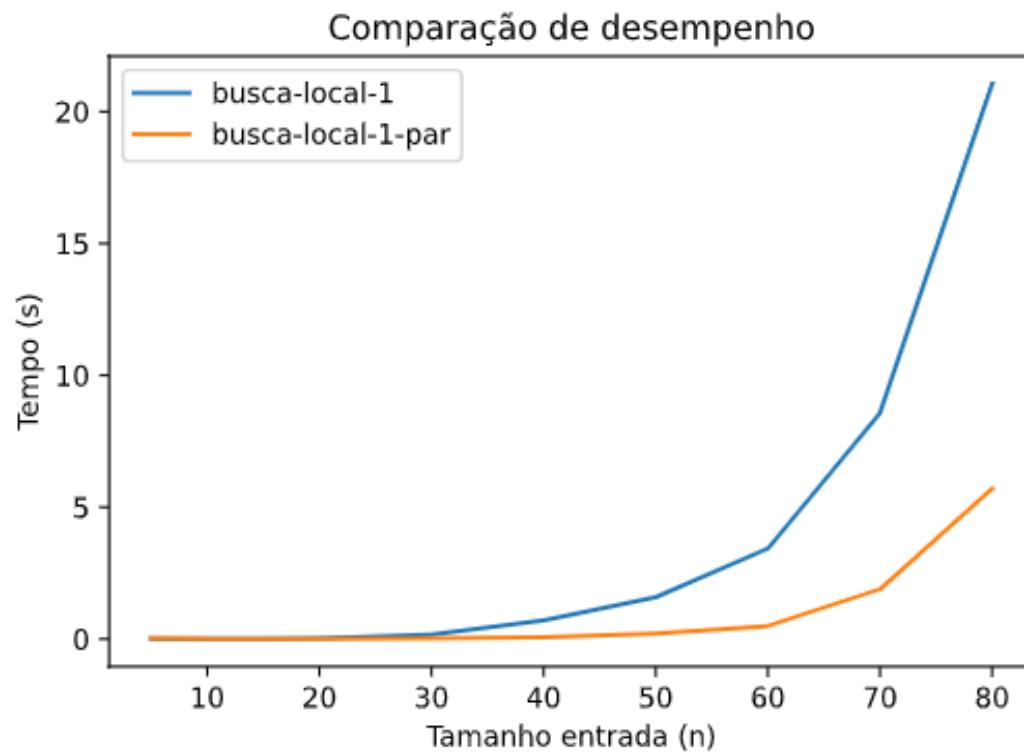




Discussão

- **Discussão 2: o quanto paralelismo faz diferença?**

Discussão 2 – Paralelismo importa

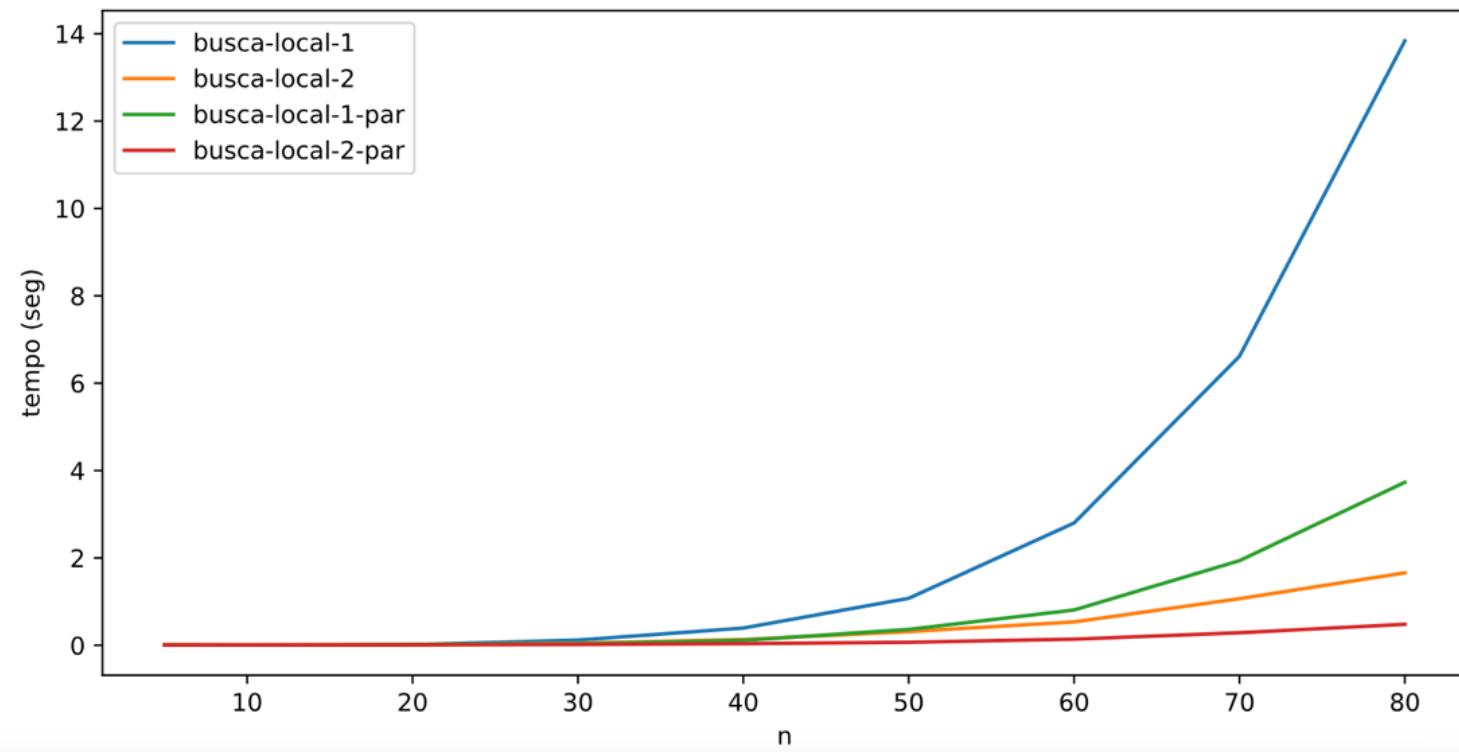




Discussão

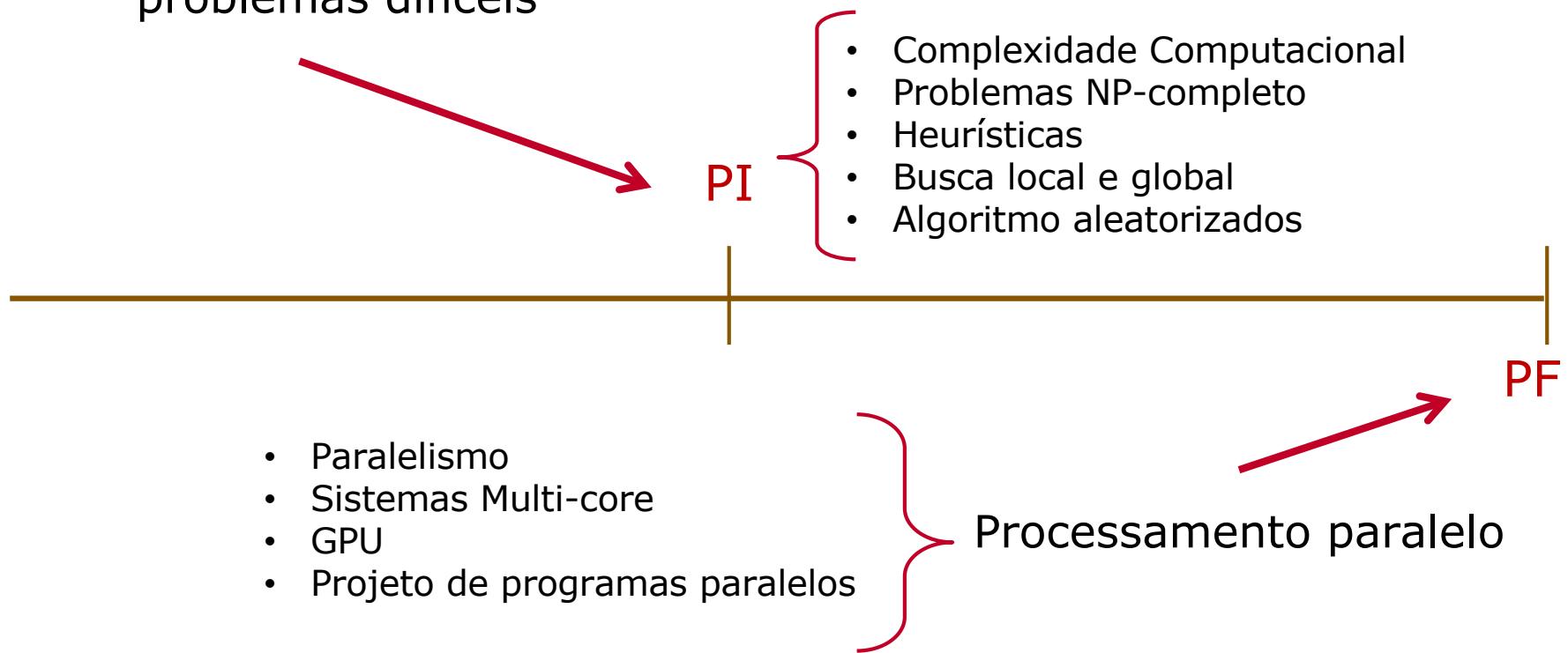
- **Discussão 3: paralelizar é a solução para algoritmos ruins?**

Paralelizar nem sempre é a (única) solução



Visão Geral da Disciplina

Estratégias para resolução de
problemas difíceis





Obrigado



Insper Supercomputação

Aulas 02/03

- Programação em C++

```
Enter rows and columns for second matrix:  
Enter elements of first matrix.  
Enter elements of matrix 1:" << endl;  
(i, j, a[i][j])  
Enter element a[" << i + 1 << j + 1 << " :  
Enter elements of matrix 2:" << endl;  
i * 1 << j * 1
```

Algoritmo

- Sequência finita de passos executáveis que resolve um problema

Algoritmo

- Sequência finita de passos executáveis que resolve um problema
- Implementar um algoritmo:
 - Transformação de um algoritmo em um programa executável

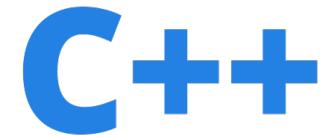
Quanto tempo um programa demora?

- Algoritmo
 - Complexidade computacional (classe de algoritmos)
 - Estruturas de Dados
- Implementação
 - Medido em segundos, para uma certa entrada
 - Tecnologias usadas (linguagens de programação, bibliotecas)
 - Hardware utilizado (clock de CPU, RAM, cache, # de núcleos, etc.)

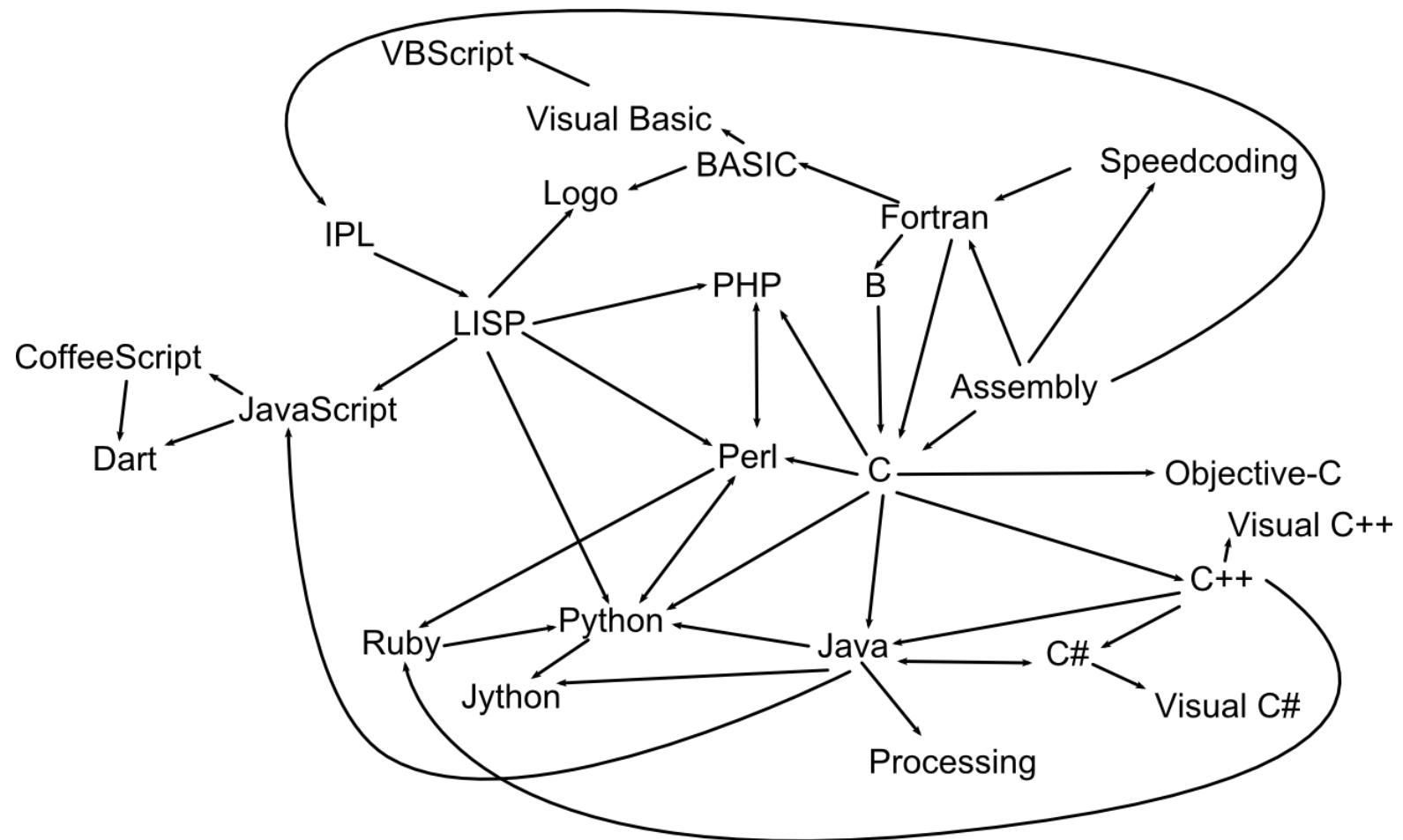
Quanto tempo um programa demora?

- Supercomputação começa quando desafios de programação acaba
- Dado um “bom” algoritmo, vamos definir:
 - Linguagem de programação adequada
 - Paralelismo indicado
 - Implementação paralela eficiente

Linguagem C++

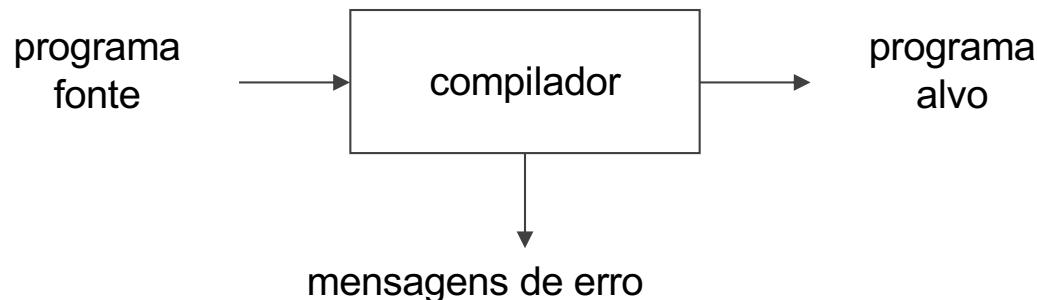


- Linguagem de programação criada no Bell Labs, em 1985
- É uma das linguagens de programação mais usadas no mundo
- C++ é derivado da linguagem C, contudo possui uma série de recursos adicionais que traz grandes vantagens para o desenvolvedor



O processo de compilação

- Compilador é um software que lê a especificação de um programa em uma linguagem-fonte o traduz em um programa em uma linguagem-alvo



O processo de compilação

- Para compilar um programa em C++ você pode usar a chamada g++

```
$ g++ your_file.cpp -o your_program
```

- Se você estiver no MacOS, uma possível linha para compilar é:

```
$ clang++ -std=c++11 -stdlib=libc++ -Wall hello.cpp -o hello
```

C++, Hello World

- Um programa “Hello World” em C++ é muito parecido com um em C.

```
#include <iostream>

int main() {
    std::cout << "Hello World!\n";
}
```

Pontos comuns com C

- Função principal: int main(int argc, char *argv[]);
- Comentários: /* */ //
- Tipos de dados: int, float, double, char
- Variações de dados: unsigned, short, long
- Qualificadores de variáveis: const, static
- Casting: (int) (float) (char)
- Operações: +, -, *, /, %
- Atribuição: =, +=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=
- Incremento e decremento: ++ --
- Operadores lógicos: !, &&, ||
- Operador condicional ternário : (? :)
- Operador vírgula: (,)

Pontos comuns com C

- Operadores Bitwise : (&, |, ^, ~, <<, >>)
- Construção Condicional : if, else, switch
- Loops: while, do-while, for
- Comparadores: ==, !=, >, <, >=, <=
- Diretivas de pré-processamento (#define, etc.)
- Declaração de funções: type func(...) { ... }
- Vetores e Matrizes: type name [elements][...];
- Enumeradores: enum
- Organização de dados: structs
- Redefinidor de tipos: typedef

C++ - Input/Output (Streams)

- C++ utiliza uma abstração conveniente denominada Streams para executar a entrada e saída de dados.

stream	description
cin	standard input stream
cout	standard output stream
cerr	standard error (output) stream
clog	standard logging (output) stream

C++ - Saída de dados (iostream)

- Utilize o std::cout com dois sinais de menor (<<) para indicar que quer enviar os dados para o console.

```
std::cout << "Bom dia" << std::endl;
```

- std::cout aceita qualquer tipo de dados.

C++ - Entrada de dados (iostream)

- Use o std::cin com dois sinais de maior (>>) para indicar que quer capturar os dados do console

```
string texto;  
std::cin >> texto;
```

C++ - Namespaces

- O namespace declara uma região de escopo para os identificadores
- Exemplo de uso de namespace:

```
std::cout << "Ola";
```

- Outra possibilidade:

```
using namespace std;  
cout << "Ola";
```

C++ - Inicialização de variáveis

- Tradicionalmente

```
type identifier = initial_value;
```

- Inicialização com valor inicial

```
type identifier (initial_value);  
int x (0);
```

- Ou

```
type identifier {initial_value};  
int x {0};
```

C++ - Inicialização Vetores/Matrizes

- C++ permite preencher um vetor sem definir seu tamanho, se você deixar os colchetes vazios []
- Nesse caso, o compilador assumirá automaticamente o tamanho para o vetor que corresponde ao número de valores incluídos entre as chaves {}.

```
int foo [] = { 16, 2, 77, 40, 12071 };
```

Tipos de dados

- Principais novos tipos de dados que foram introduzidos em C++
 - bool: true / false
 - string: armazena textos e possui recursos para tratar textos

```
#include <string>
string texto;
texto = "exemplo de texto";
std::cout << texto << std::endl;
```



Atividade prática

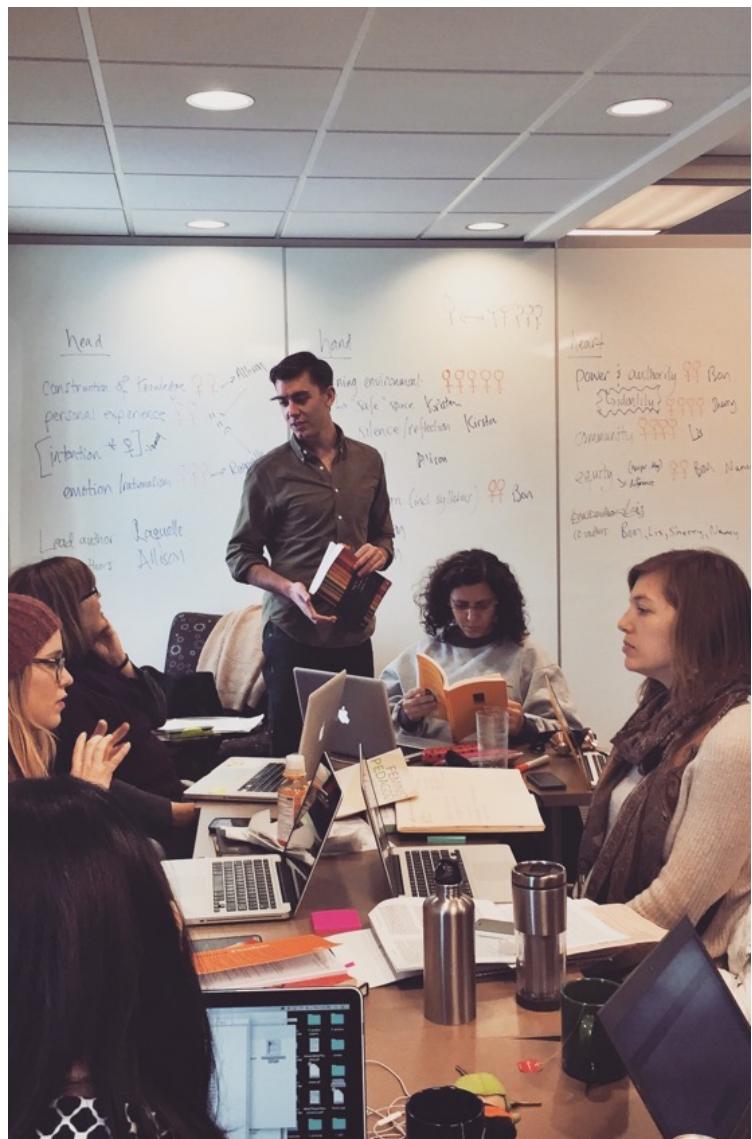
Nesta aula iremos implementar códigos em C++ tratando os seguintes aspectos:

- estrutura básica
- variáveis
- condicionais
- laços
- vector
- funções – passagem por valor e passagem por referência

Iremos desenvolver código em tempo real.

Aproveite para relembrar a lógica de programação necessária

E foque no algoritmo que foi escolhido par resolver a situação problema.



Material - Professor

- Códigos-fonte desenvolvidos em conjunto com alunos na aula 02
- O roteiro que está na página é utilizado na aula 03 para os alunos consolidarem os conceitos aqui desenvolvidos

01. Estrutura

```
#include<iostream>

int main() {
    std::cout << "Hello, World!\n";
    std::cout << "2022\n";
    return 0;
}
```

02. Variáveis

```
1 #include<iostream>
2 #include<string>
3 using namespace std;
4 int main() {
5
6     int x;
7     cout << "Entre com um número: ";
8     cin >> x;
9
10    cout << "Você digitou o valor " << x << endl;
11
12    int y;
13    string s;
14    y = 2*x;
15    x = y + 5;
16    cout << "x = " << x << endl;
17    cout << "y = " << y << endl;
18
19    cout << "Digite uma string" << endl;
20    cin >> s;
21    cout << "Você digitou " << s << endl;
22
23    return 0;
24 }
```

03. Condicionais

```
1 #include<iostream>
2 using namespace std;
3
4 int main() {
5     int x;
6     cout << "Entre com um número: ";
7     cin >> x;
8
9     if(!cin) {
10         cout << "Você não digitou um número!" << endl;
11         return 1;
12     }
13     int y;
14     cout << "Entre com outro número: ";
15     if(!(cin >> y)){
16         cout << "Você não digitou um número!" << endl;
17         return 1;
18     }
19
20     if (x < y){
21         cout << "O primeiro número é menor que o segundo" << endl;
22     }
23     else if (x == y) {
24         cout << "Os dois números são iguais" << endl;;
25     }
26     else {
27         cout << "O primeiro número é maior que o segundo" << endl;
28     }
29
30     cout << "Digite dois números" << endl;
31     cin >> x >> y;
32     if(!cin){
33         cout << "Alguma coisa saiu errado!" << endl;
34     }
35
36
37
38
39
40     return 0;
41 }
```

04. Laços e Vetores

```
1 #include <iostream>
2 #include <vector>
3 using namespace std;
4 int main()
5 {
6     int x;
7     int w[] = {1, 2, 3};
8     cout << w[0] << endl;
9
10    vector<int> v;
11    cout << "Entre quantos numeros desejar:" << endl;
12    while (cin >> x)
13    {
14        v.push_back(x);
15    }
16
17    size_t i = 0;
18    size_t greatest_index = 0;
19    size_t least_index = 0;
20    while (i < v.size())
21    {
22        if (v[i] > v[greatest_index])
23            greatest_index = i;
24        if (v[i] < v[least_index])
25            least_index = i;
26        i += 1;
27    }
28
29    if (v.empty())
30    {
31        cout << "O vetor é vazio." << endl;
32    }
33    else
34    {
35        cout << "Maior elemento: " << v[greatest_index] << endl;
36        cout << "Menor elemento: " << v[least_index] << endl;
37    }
38
39    return 0;
40 }
```

05. Funções e Vector

```
1 #include <iostream>
2 #include <vector>
3 #include <string>
4 using namespace std;
5
6 vector<int> read_int_vector();
7 float average(vector<int> v);
8 int sum(vector<int> v);
9
10 int main() {
11
12     vector<int> v = read_int_vector();
13     // for(auto e: v)
14     //     cout << e << endl;
15     cout << "Média " << average(v) << endl;
16
17     return 0;
18 }
19
20 vector<int> read_int_vector() {
21     vector<int> result;
22     int x;
23     cout << "Entre com os números:" << endl;
24     while (true) {
25         while(cin >> x)
26             result.push_back(x);
27         if(cin.eof())
28             break;
29         cin.clear();
30         string s;
31         getline(cin, s);
32         cout << "Atenção, ignorando: " << s << endl;
33     }
34
35     return result;
36 }
37
38 float average(vector<int> v){
39     if(v.empty())
40         return 0;
41     int size = v.size();
42     return sum(v)/size;
43 }
44
45
46 int sum(vector<int> v) {
47     int total = 0;
48     for(auto e: v)
49         total += e;
50     cout << "total = " << total << endl;
51     return total;
52 }
```

06. Funções / Valor

```
1 #include<iostream>
2 using namespace std;
3
4 int sqr_it(int x);
5
6 int main() {
7     int t = 10;
8     cout << sqr_it(t) << endl;
9     return 0;
10 }
11
12 int sqr_it(int x){
13     return x*x;
14 }
15
```

07. Funções / Referência

```
1 #include<iostream>
2 using namespace std;
3
4 int sqr_it(int& x);
5
6 int main() {
7     int t = 10;
8     cout << sqr_it(t) << endl;
9     return 0;
10 }
11
12 int sqr_it(int& x){
13     return x*x;
14 }
15
```

08. Vector

```
1 #include<iostream>
2 #include<vector>
3 #include<algorithm>
4
5 using namespace std;
6 bool my_compare(float a, float b);
7 int main() {
8
9     vector<float> nums;
10
11    //inserindo um valor no vector
12    nums.push_back(10.5);
13
14    //obtendo o valor
15    cout << nums[0] << endl;
16
17    //apagando o primeiro elemento
18
19    nums.erase(nums.begin());
20
21    //exibindo o tamanho do vector
22
23    cout << nums.size() << endl;
24
25    nums.push_back(4.0);
26    nums.push_back(7.5);
27    nums.push_back(23.4);
28
29    cout << nums.size() << endl;
30
31    //fazendo uso de assign, popular um vector
32    nums.assign(100, 0.5); //adiciona 100 floats de valor 0.5
33
34    cout << nums.size() << endl;
35
36    for(auto& e: nums)
37        cout << e << "\t";
38
39    //apagando o vector
40    nums.clear();
41
42    nums.push_back(4.0);
43    nums.push_back(7.5);
44    nums.push_back(23.4);
45
46    //ordenando
47    sort(nums.begin(), nums.end(), my_compare);
48
49    for(auto e: nums)
50        cout << e << "\t";
51
52
53
54
55    return 0;
56 }
57
58 bool my_compare(float a, float b){
59     return a > b; //ordenacao decrescente
60 }
```



Obrigado



Insper Supercomputação

Aula 04

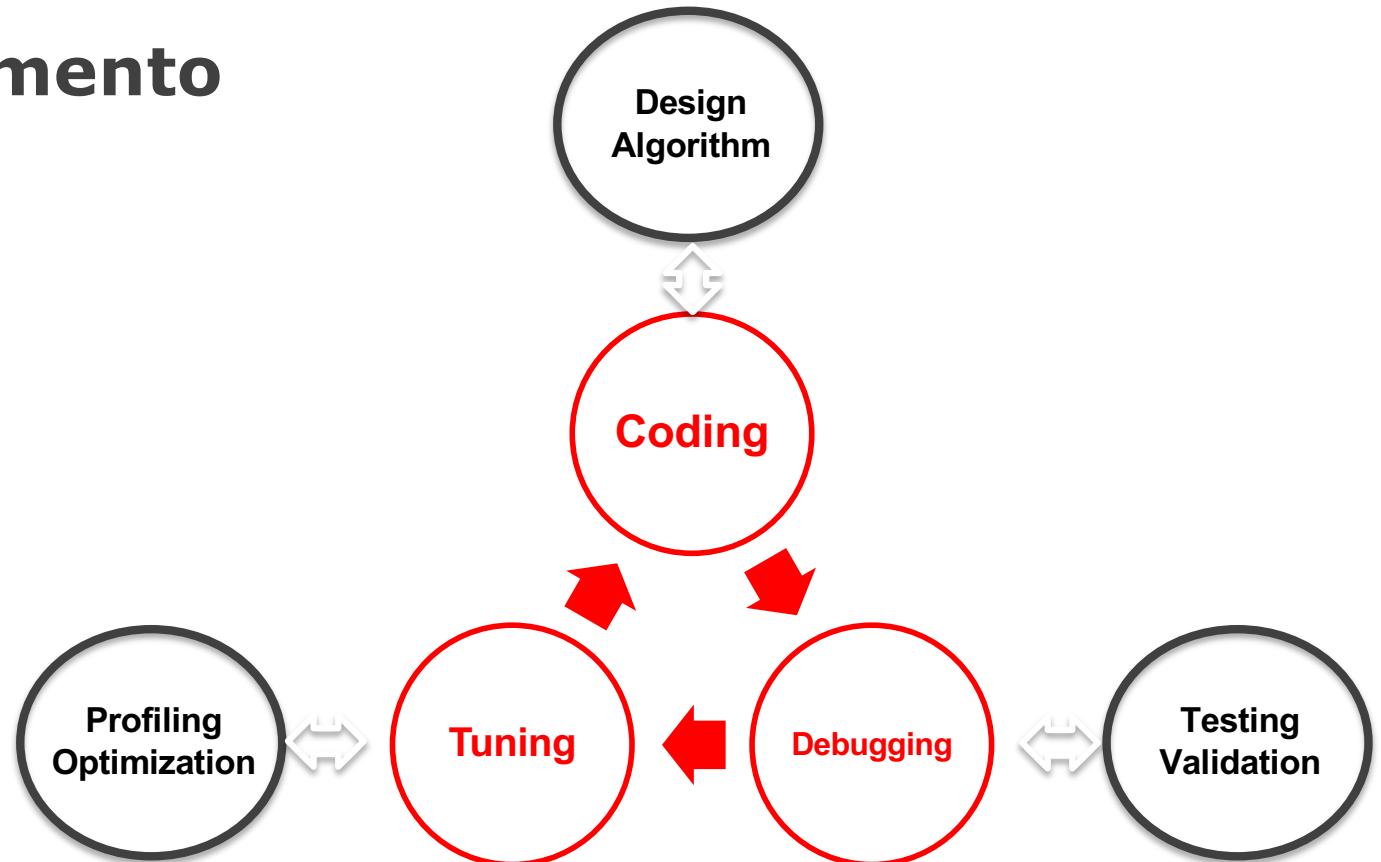
- Profiling



Vamos recordar...

- Soluções de alto desempenho se dão normalmente sob três estratégias:
 1. Algoritmos eficientes
 2. Implementação Eficiente
 1. Cache, paralelismo de instrução
 2. Linguagem de Programação adequada
 3. Paralelismo

Ciclo habitual de desenvolvimento



S.Y. Jun (SCD/PDS) | Profiling Tutorial | LArSoft Workshop



Medição de desempenho

- Otimizamos um algoritmo:
 - Como medir quanto tempo cada função demora?
 - Nossa função ficou mais rápida? Se sim, quanto? Se não, por quê?
 - Como medir “quantidade de trabalho feito”?

Profiling

- Análise de um programa durante sua execução, de modo a determinar seu consumo de memória e/ou tempo de execução
- Com profiling, podemos responder duas importantes perguntas:
 - Onde o programa consome mais recursos?
 - Onde devo concentrar meus esforços de otimização?



Warm-up (roteiro)

- O problema de soma de uma matriz
- No roteiro da aula, é apresentado um código-fonte em C++ de suas funções: **naive_sum** e **improved_sum**

Vamos fazer uso da ferramenta **Valgrind** para realizar o profiling desse código e entender a diferença entre as duas funções

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

constexpr int M = 2048;
constexpr int N = 2048;

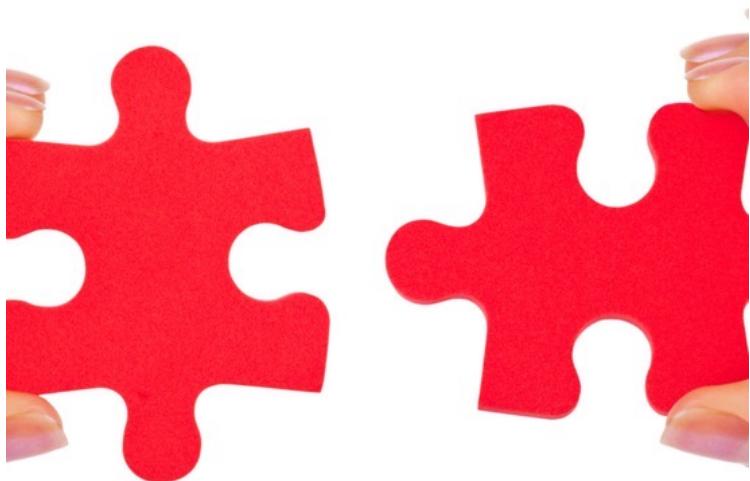
double naive_sum(const double a[][N]){
    double sum = 0.0;
    for(int j = 0; j < N; ++j) {
        for(int i = 0; i < M; ++i)
            sum += a[i][j];
    }
    return sum;
}

double improved_sum(const double a[][N]) {
    double sum = 0.0;
    for(int i = 0; i < M; ++i)
        for(int j = 0; j < N; ++j)
            sum += a[i][j];
    return sum;
}

int main() {
    static double a[M][N];
    fill_n(&a[0][0], M*N, 1.0 / (M*N));
    cout << naive_sum(a) << endl;
    static double b[M][N];
    fill_n(&b[0][0], M*N, 1.0 / (M*N));
    cout << improved_sum(b) << endl;
    return 0;
}
```

Roteiro

- Vamos desenvolver o roteiro da aula e conhecer as ferramentas disponíveis no Valgrind para proling de dados



Conclusões

- Entrada e saída custam caro
- Implementações diferentes do mesmo algoritmo podem ter desempenho diferentes
- Detalhes finos só são visíveis com o auxílio de ferramentas de profiling



Obrigado



Insper Supercomputação

Aula - 05

- Heurísticas e problemas difíceis
- C++/Struct
- Início do projeto

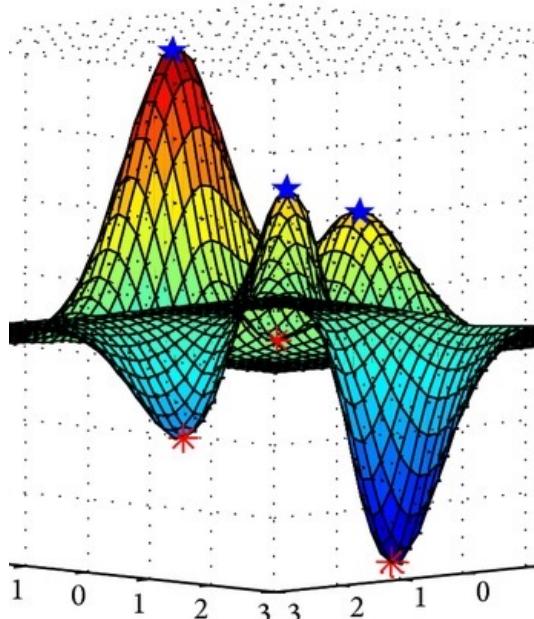
Resolução de problemas

© 2000 Randy Glasbergen.
www.glasbergen.com



- Problemas difíceis aparecem em muitas áreas
 - Pesquisa operacional (logística, produção, etc.)
 - Machine Learning
 - Marketing
 - Planejamento Urbano - Mobilidade

Resolução de Problemas - Otimização



- Função objetivo
 - Algo que queremos maximizar ou minimizar
- Restrições
 - Definem quais possíveis soluções são válidas
- Muitas classes de problemas:
 - Programação Linear / Inteira
 - Programação não-linear
 - Otimização combinatória

Otimização combinatória

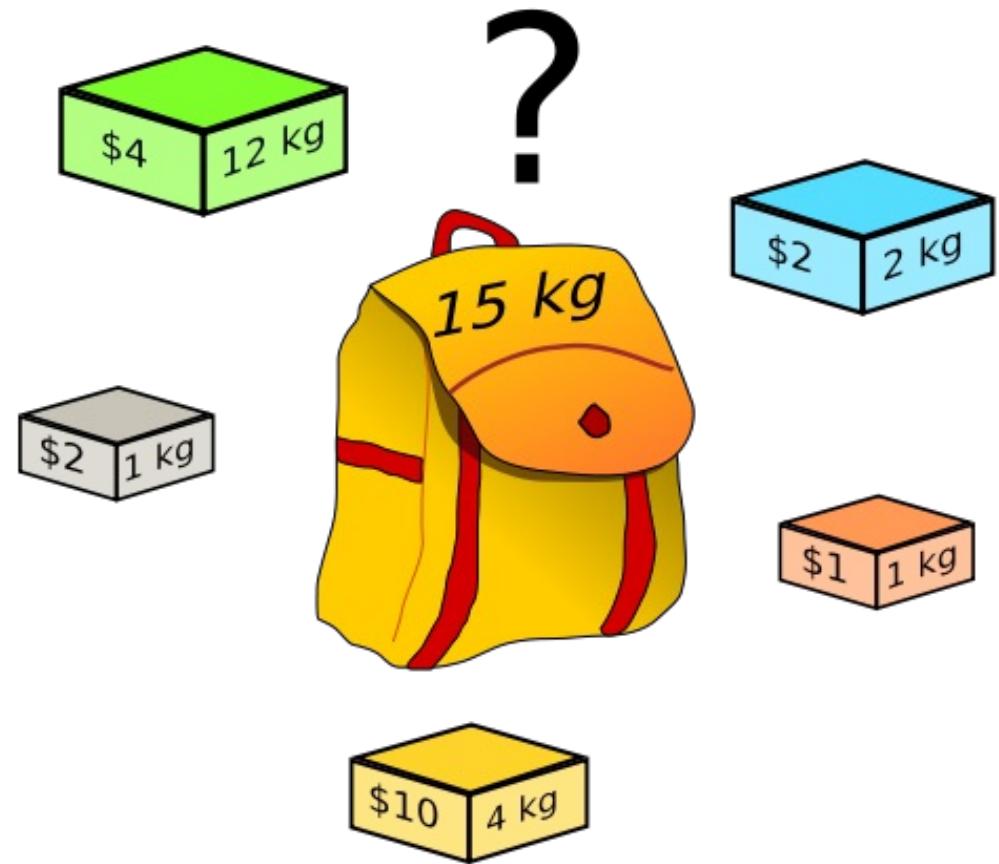
- Tem por objetivo selecionar um objeto com melhor função objetivo dentre uma coleção finita
- Não tem derivada
- Não tem vizinhança
- Coleção não é densa
- Técnicas tradicionais de cálculo e otimização não funcionam, pois nosso problema é **discreto**

Nosso problema: A mochila binária



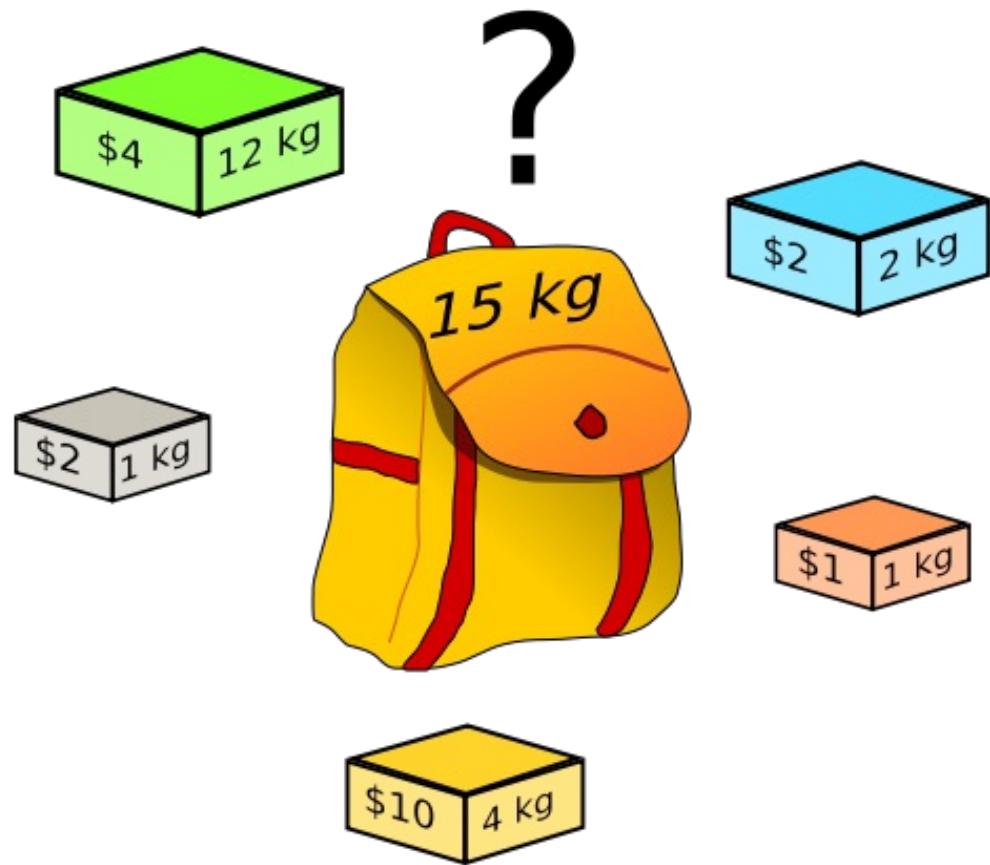
Civil War Knapsack. U.S. government image. Vicksburg National Military Park. Public domain.

A mochila binária



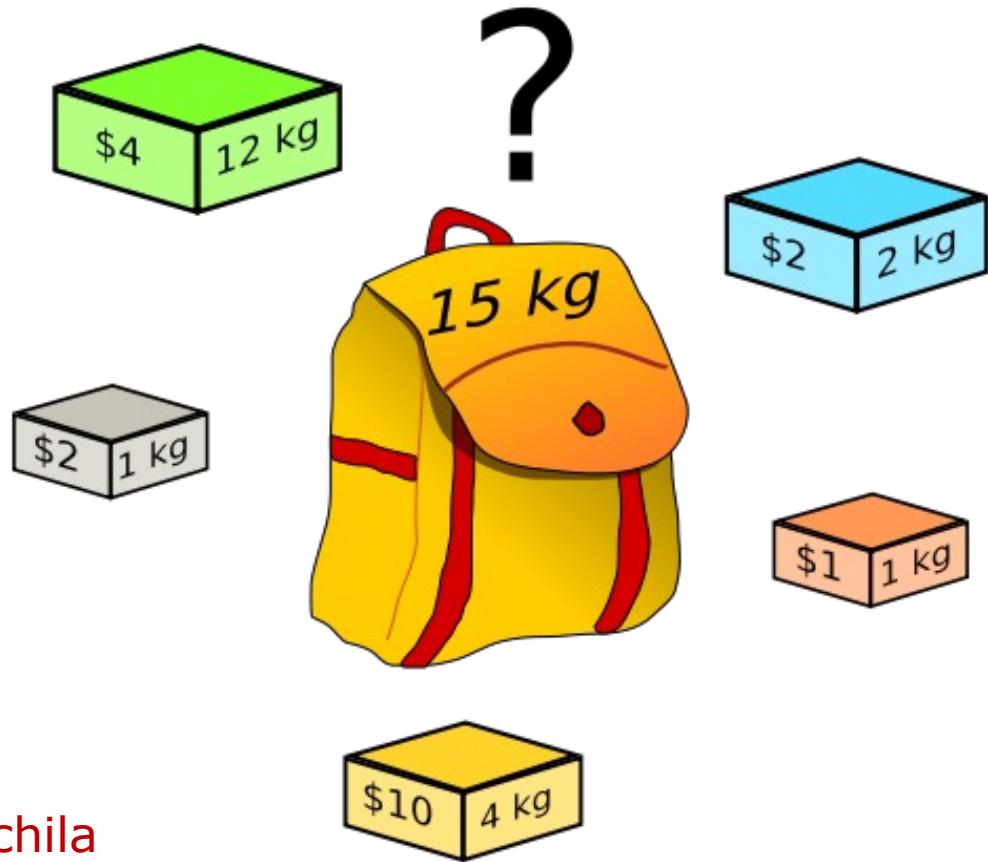
A mochila binária

- Quais escolhas podem ser feitas?
- Qual é a função objetivo?
- Quais são as restrições?



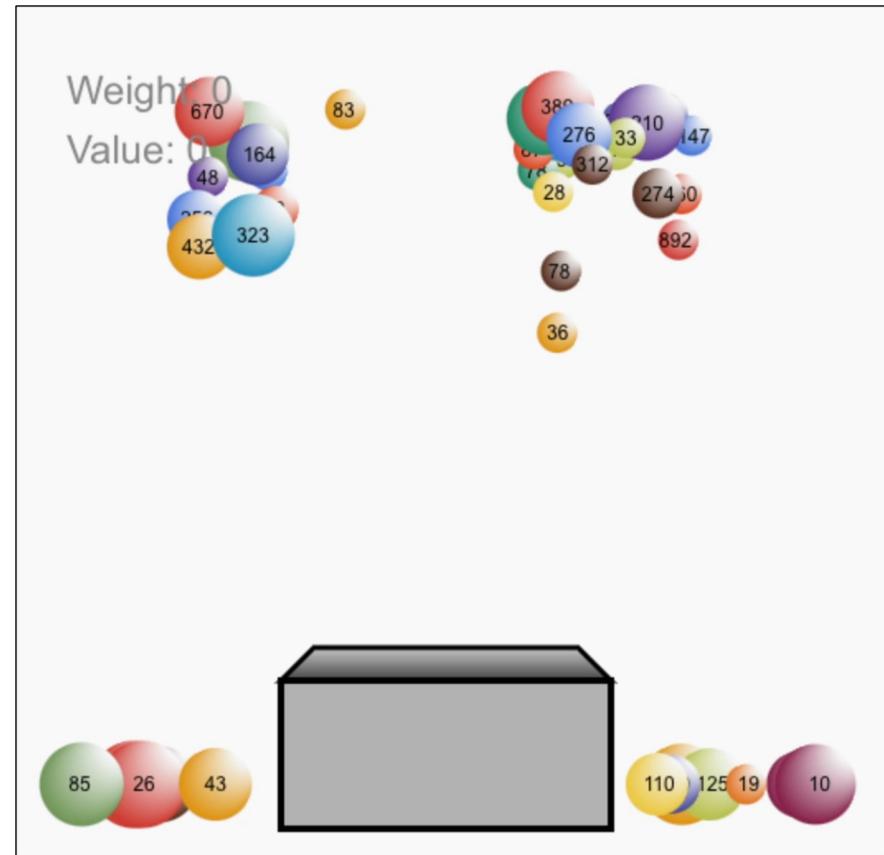
A mochila binária

- Quais escolhas podem ser feitas?
 - Quais produtos pegar?
- Qual é a função objetivo?
 - Maximizar valor os objetos capturados
- Quais são as restrições?
 - Peso dos objetos não pode exceder a capacidade da mochila



A mochila binária

Na animação ao lado, 50 itens são colocados em uma mochila. Cada item tem um valor (o número no item) e um peso (aproximadamente proporcional à área do item). A mochila é tem capacidade de 850, e nosso objetivo é encontrar o conjunto de itens que irão maximizar o valor total sem exceder a capacidade.



Fonte: <https://developers.google.com/optimization/bin/knapsack>



Como resolver esse problema?

- Algumas opções:
 - Tentar tudo e ver qual é melhor
 - Pegar o mais caro primeiro
 - Pegar o mais leve primeiro
- É possível resolver de maneira eficiente?



Como resolver esse problema?

- Algumas opções:
 - Tentar tudo e ver qual é melhor
 - Pegar o mais caro primeiro
 - Pegar o mais leve primeiro
- É possível resolver de maneira eficiente?
NÃO

Heurística



- “truque” usado para resolver um problema rapidamente
- Por velocidade, sacrificamos ao menos um entre:
 - Optimalidade
 - Corretude
 - Precisão
 - Exatidão

Heurística



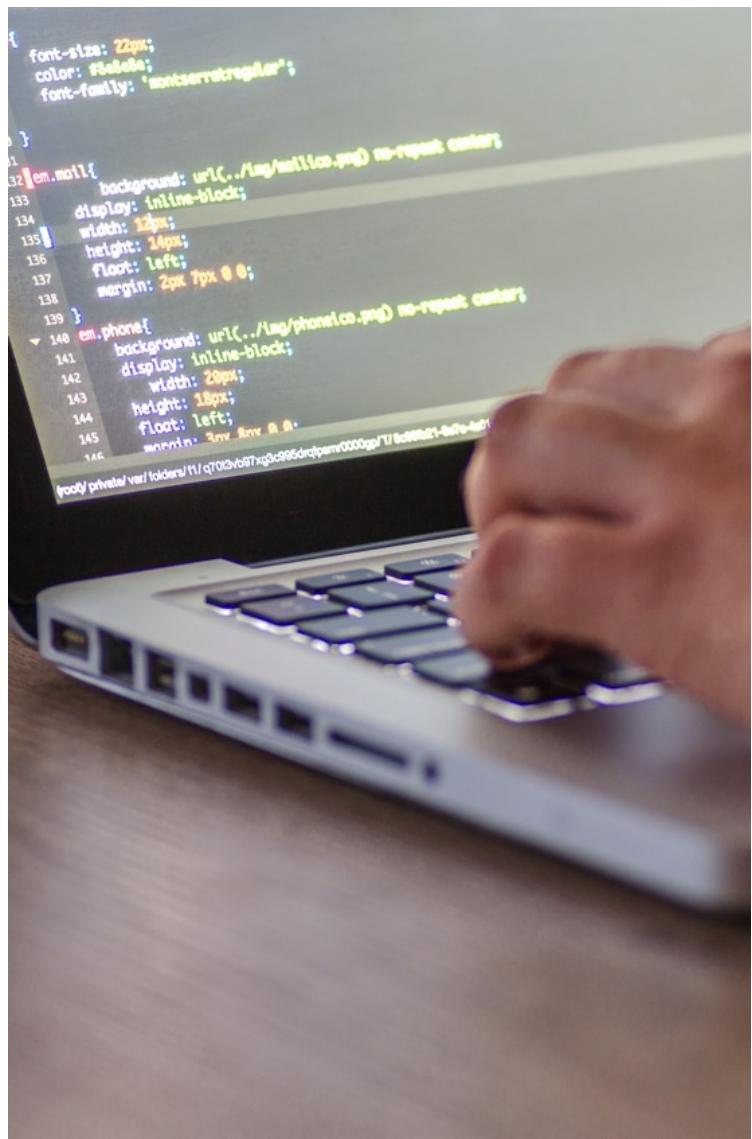
- Uma boa heurística é suficiente para obter resultados aproximados ou ganhos de curto prazo
- Processo:
 - Explorar alguma propriedade do problema
 - Dividir em partes menores, que podem ser resolvidas rapidamente e combinar os resultados

Heurística para a mochila



Civil War Knapsack. U.S. government image. Vicksburg National Military Park. Public domain.

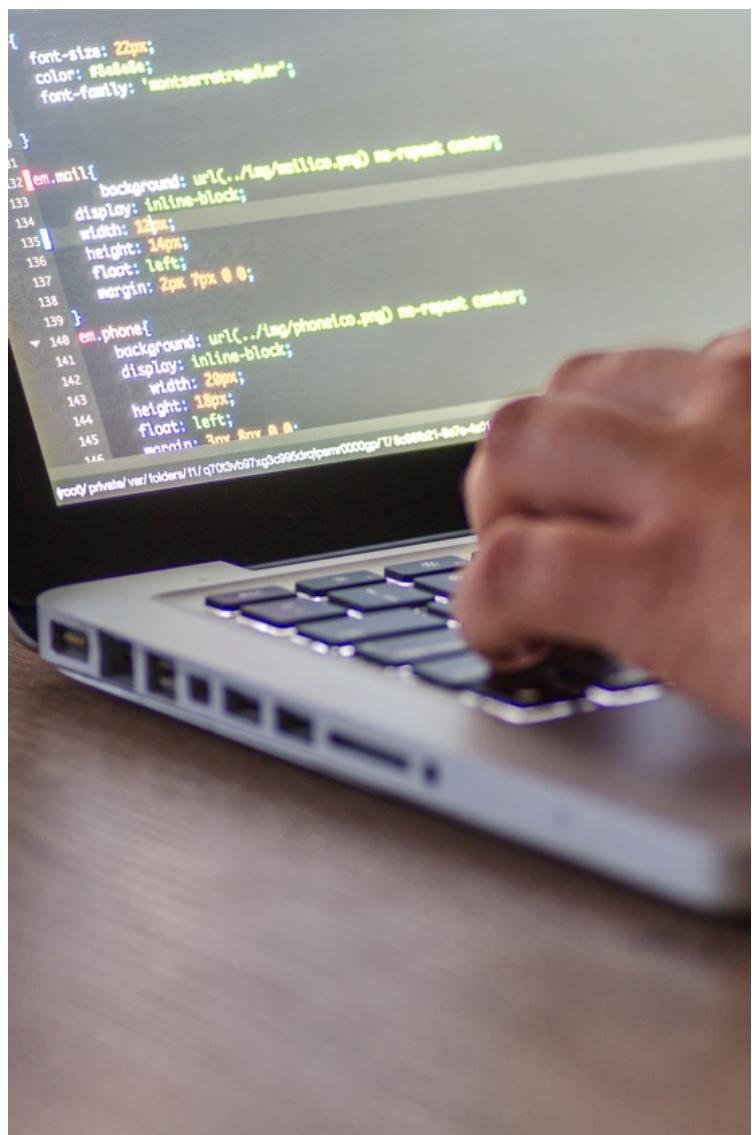
- Algumas opções:
- Pegar **o mais caro** primeiro
- Pegar **o mais leve** primeiro



Atividade prática

Resolvendo a mochila binária

Sua missão: implementar as duas heurísticas e comparar seus resultados



Dica: C++ Struct

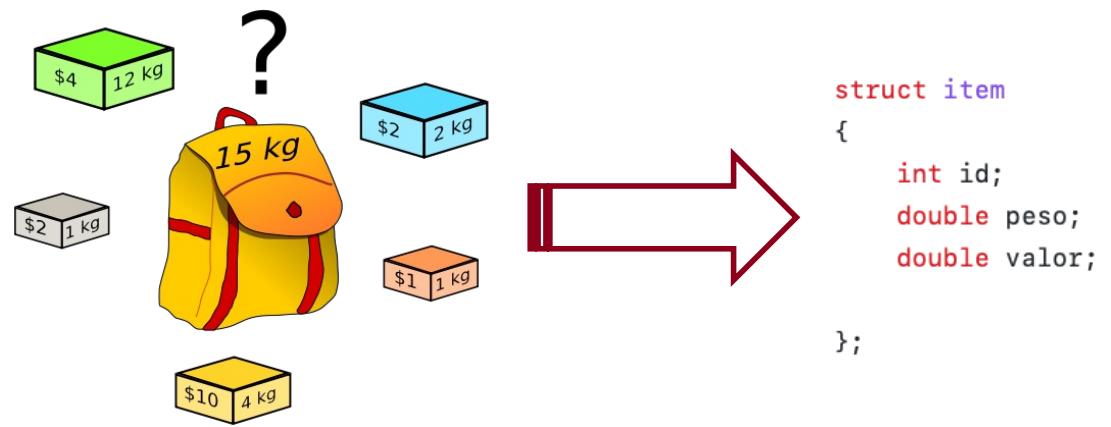
Uma boa abordagem para modelar uma mochila que é fazer uso de vector. Além disso, os itens (que possuem peso e valor) podem ser criados como structs em c++

```
#include <iostream> 1
using namespace std; 2
struct Person 3
{
    int citizenship; 5
    int age; 6
}; 7
int main(void) { 8
    struct Person p; 9
    p.citizenship = 1; 10
    p.age = 27; 11
    cout << "Person citizenship: " << p.citizenship << endl; 12
    cout << "Person age: " << p.age << endl; 13
    return 0; 14
} 15
```



Dica: C++ Struct

Uma boa abordagem para modelar uma mochila que é fazer uso de vector. Além disso, os itens (que possuem peso e valor) podem ser criados como structs em c++





Discussão

- Qual a complexidade computacional das abordagens?
- Quando uma é melhor que a outra?
- Alguma consegue obter o melhor valor possível?



Obrigado



Insper Supercomputação

Aula - 06

- Exploration / Exploitation
- Resolução da mochila por algoritmos genéticos
- Algoritmos aleatorizados

Heurísticas

- Vimos que heurísticas são “truques” usados para resolver um problema rapidamente
- Uma boa heurística consegue obter resultados aproximados ou ganhos de curto prazo, porém não garante resultados ótimos, nem resultados bons em todas as situações!

Heurísticas - Limitações

- E se a solução gerada não for boa? Conseguimos “tentar” de novo e gerar outras parecidas?
- Será que é possível melhorar a solução gerada? Como?

Exploration vs. Exploitation



Exploration

Exploration:

- decisão não localmente ótima feita "de propósito"
- visa adicionar variabilidade nas soluções geradas



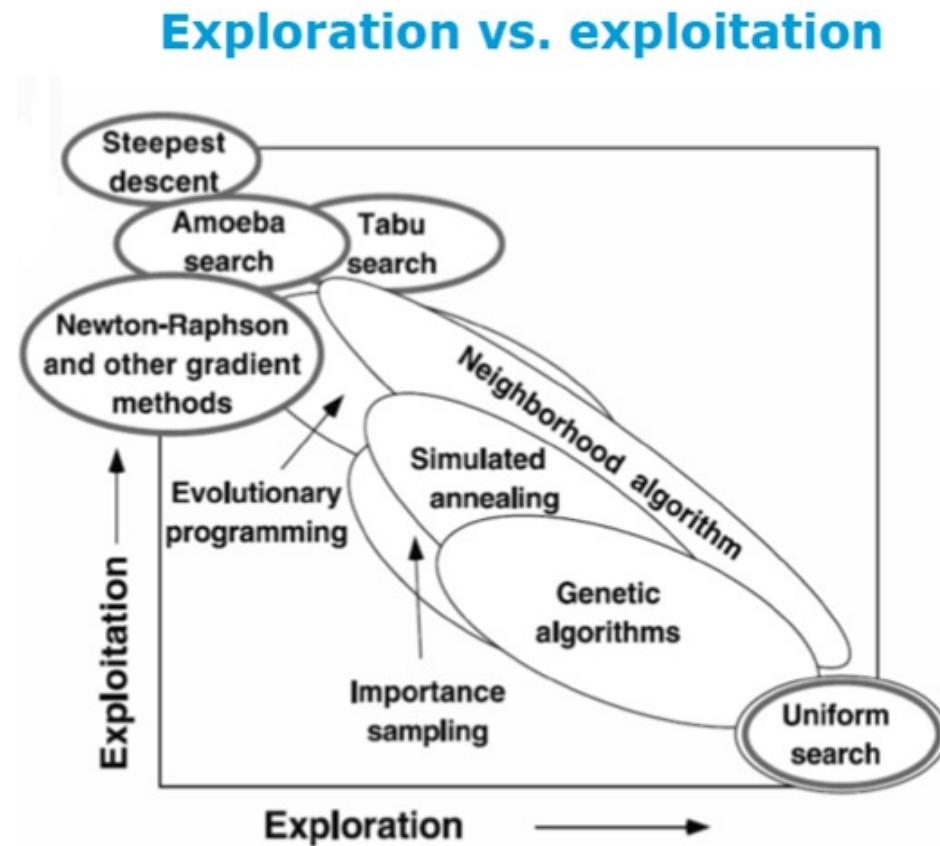
Exploitation

Exploitation:

- explorar alguma **propriedade do problema**
- pode ser uma intuição que leve a bons resultados em curto prazo

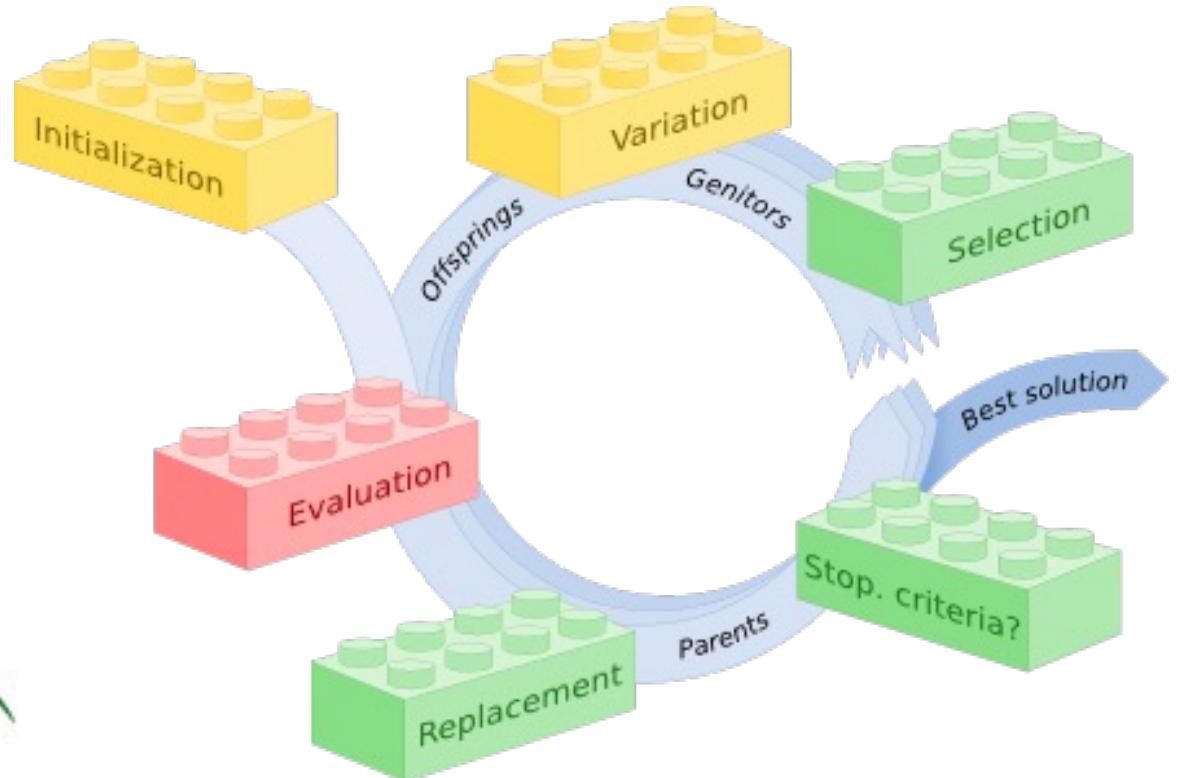
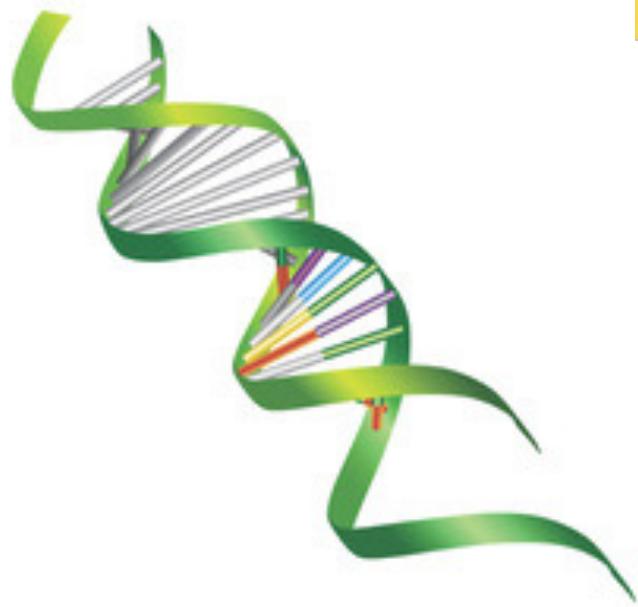
Exploration vs. Exploitation

- Possíveis técnicas.
Você reconhece alguma?



picture courtesy of Heriot-Watt university

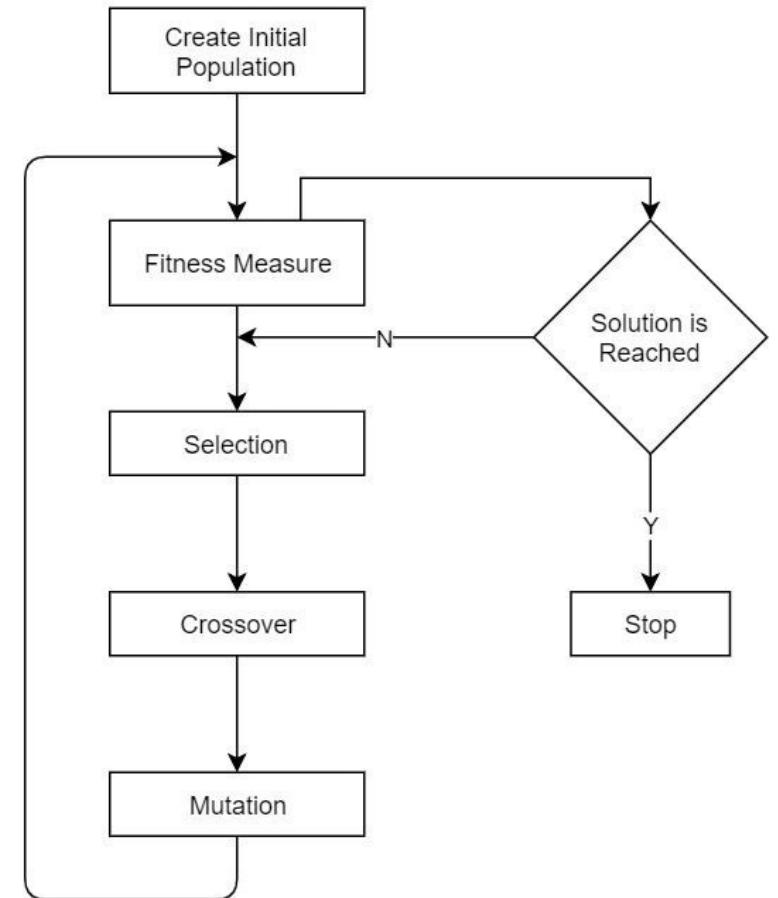
Algoritmos genéticos



Algoritmo Genético para a Mochila Binária

Exploration vs Exploitation em Algoritmos Genéticos

- Estratégia de Reprodução (cross-over)
- Taxa de mutação
- Estratégia de elitismo



Possíveis itens:		
Item	Peso	Valor
1	12	32
2	2	29
3	1	29
4	5	4
5	6	2
6	4	20
7	7	34
8	1	39
9	1	48
10	2	44

Capacidade da mochila: 35

População Inicial
(n=8)

$$fitness = \sum_{i=1}^n c_i v_i; \text{ if } \sum_{i=1}^n c_i w_i \leq kw$$

$$fitness = 0; \text{ otherwise}$$

Mochila Binária por Algoritmos Genéticos



Tamanho da população = (8, 10)

População inicial:

```
[[1 0 1 1 0 0 0 1 1 1]
 [1 0 0 0 0 1 1 0 1 1]
 [0 0 0 0 1 0 1 1 1 1]
 [0 0 0 0 0 1 0 1 0 0]
 [1 1 1 1 0 0 1 0 1 0]
 [1 0 0 0 1 1 1 1 0 0]
 [1 1 1 0 0 0 1 0 0 0]
 [0 1 1 0 1 1 0 1 0 1]]
```

```
array([[196],
 [178],
 [167],
 [59],
 [176],
 [127],
 [124],
 [163]])
```

Fitness

Seleção

```
[1 0 1 1 0 0 0 1 1 1]
 [0 0 0 0 1 0 1 1 1 1]
```

```
[1 0 0 0 0 1 1 0 1 1]
 [1 1 1 1 0 0 1 0 1 0]
```

Cross-OVER

```
[1 0 1 1 0 | 0 0 1 1 1]
 [1 0 0 0 0 | 1 1 0 1 1]
```

```
[1 0 1 1 0 1 1 0 1 1]
 [1 0 0 0 0 0 0 1 1 1]
```

Mutação

```
[1 0 0 0 0 0 0 1 1 1]
```

```
[1 0 0 0 1 0 0 1 1 1]
```

Repetir
por N
gerações

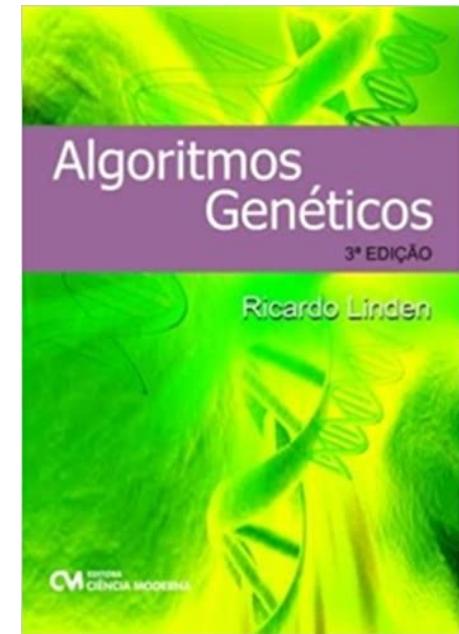
Implementação em Python

- Clique [aqui](#) para conhecer a implementação em Python para a mochila binária por meio de algoritmos genéticos
- Atenção:
- 0,5 ponto extra na nota final se entregar a implementação em C++



Para saber mais ...

- <https://www.algoritmosgeneticos.com.br>



Extra ... No mundo real ...

Automated Antenna Design with Evolutionary Algorithms

Gregory S. Hornby* and Al Globus

University of California Santa Cruz, Mailtop 269-3, NASA Ames Research Center, Moffett Field, CA

Derek S. Linden

JEM Engineering, 8683 Cherry Lane, Laurel, Maryland 20707

Jason D. Lohn

NASA Ames Research Center, Mail Stop 269-1, Moffett Field, CA 94035

G. S. Hornby, J. D. Lohn, and D. S. Linden

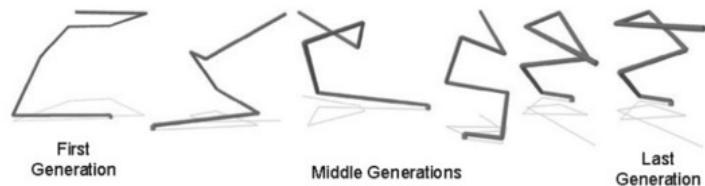


Figure 10: Sequence of evolved antennas leading up to antenna ST5-33.142.7.



De volta para a Mochila

- Nossa heurística é **100% exploitation**
- Como podemos adicionar Exploration?

De volta para a Mochila

- Nossa heurística é **100% exploitation**
- Como podemos adicionar Exploration?
 - Alternar heurísticas **de vez em quando**
 - **De vez em quando faço** uma escolha qualquer
 - Inverto a heurística **de vez em quando**

De volta para a Mochila

- Nossa heurística é **100% exploitation**
- Como podemos adicionar Exploration?
 - Alternar heurísticas **de vez em quando**
 - **De vez em quando faço** uma escolha qualquer
 - Inverto a heurística **de vez em quando**

Aleatoriedade

Exploration

- Exploration requer a capacidade de criar um programa que executa de maneira diferente a cada execução
- Precisamos:
 - 1. Criar uma fonte de aleatoriedade;
 - 2. Uma maneira de gerar sequências de números aleatórios

Números aleatórios

- Um gerador de números aleatórios é impossível de ser criado usando um computador
- 1. É impossível predizer qual será o próximo número aleatório “de verdade”;
- 2. Um computador executa uma sequência de comandos conhecidos, baseando-se em dados guardados na memória. A execução é, portanto, determinística

Números (pseudo-)aleatórios

- Gerador de números pseudo-aleatório (pRNG): algoritmo determinístico que gera sequências de números que parecem aleatórias
- 1. Deterministício: produz sempre a mesma sequência.
- 2. Sequências que parecem aleatórias: não conseguiríamos distinguir uma sequência gerada por um pRNG e uma sequência aleatória de verdade

Gerando números aleatórios

- Sorteio de números aleatórios é dado por 2 elementos:
- 1. Gerador: produz bits aleatórios a partir de um parâmetro seed. Cada seed gera uma sequência diferente de bits
- 2. Distribuição de probabilidade: gera sequência de números a partir de um conjunto de números

Entendendo (em Python)

Random Generators

```
In [1]: import numpy as np
```

```
In [2]: np.random.rand()  
Out[2]: 0.9535543896720104
```

```
In [3]: np.random.seed(123)
```

```
In [4]: np.random.rand()  
Out[4]: 0.6964691855978616  
In [5]: np.random.rand()  
Out[5]: 0.28613933495037946
```

```
In [6]: np.random.seed(123)
```

```
In [7]: np.random.rand()  
Out[7]: 0.696469185597861  
In [8]: np.random.rand()  
Out[8]: 0.28613933495037946
```

Pseudo-random numbers
Mathematical formula
Starting from a seed

Same seed: same random numbers!
Ensures "reproducibility"

numpy.random.rand

`random.rand(d0, d1, ..., dn)`

Random values in a given shape.

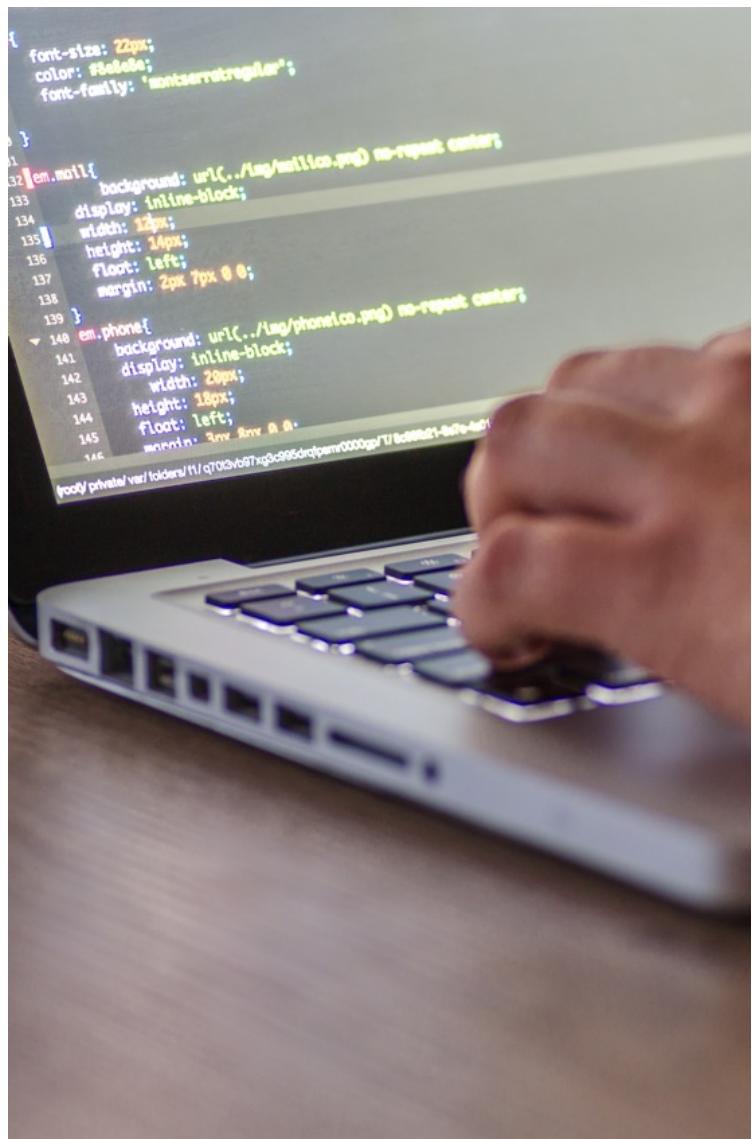
ⓘ Note

This is a convenience function for users porting code from Matlab, and wraps `random_sample`. That function takes a tuple to specify the size of the output, which is consistent with other NumPy functions like `numpy.zeros` and `numpy.ones`.

Create an array of the given shape and populate it with random samples from a uniform distribution over [0, 1].

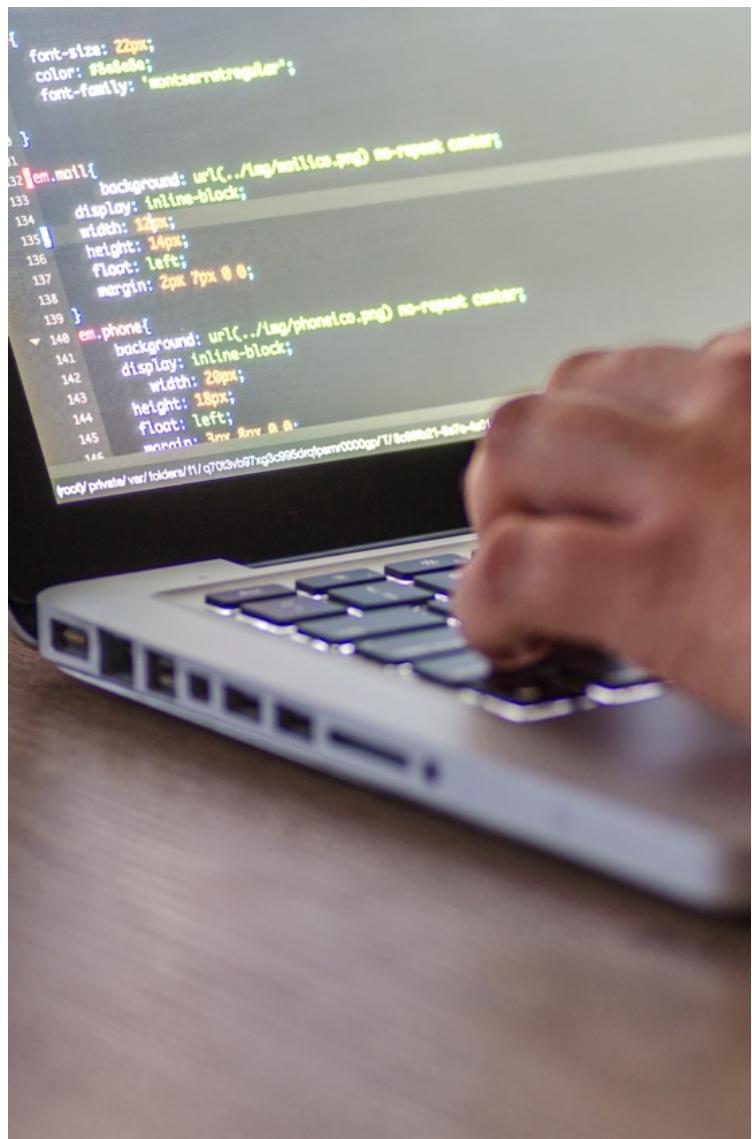
Distributions

<code>beta(a, b[, size])</code>	Draw samples from a Beta distribution.
<code>binomial(n, p[, size])</code>	Draw samples from a binomial distribution.
<code>chisquare(df[, size])</code>	Draw samples from a chi-square distribution.
<code>dirichlet(alpha[, size])</code>	Draw samples from the Dirichlet distribution.
<code>exponential([scale, size])</code>	Draw samples from an exponential distribution.
<code>f(dnum, dfden[, size])</code>	Draw samples from an F distribution.
<code>gamma(shape[, scale, size])</code>	Draw samples from a Gamma distribution.
<code>geometric(p[, size])</code>	Draw samples from the geometric distribution.
<code>gumbel([loc, scale, size])</code>	Draw samples from a Gumbel distribution.
<code>hypergeometric/ngood, nbad, nsample[, size])</code>	Draw samples from a Hypergeometric distribution.



Atividade prática 1

Resolvendo a mochila binária por meio de heurísticas com aleatoriedade



Atividade prática 2

E se tudo fosse aleatório?

Implemente uma solução completamente aleatória para a mochila binária



Discussão

- Adicionar aleatoriedade melhorou os resultados?
- Qual a qualidade das soluções aleatórias?



Obrigado



Insper Supercomputação

Aula - 07

- Busca Local

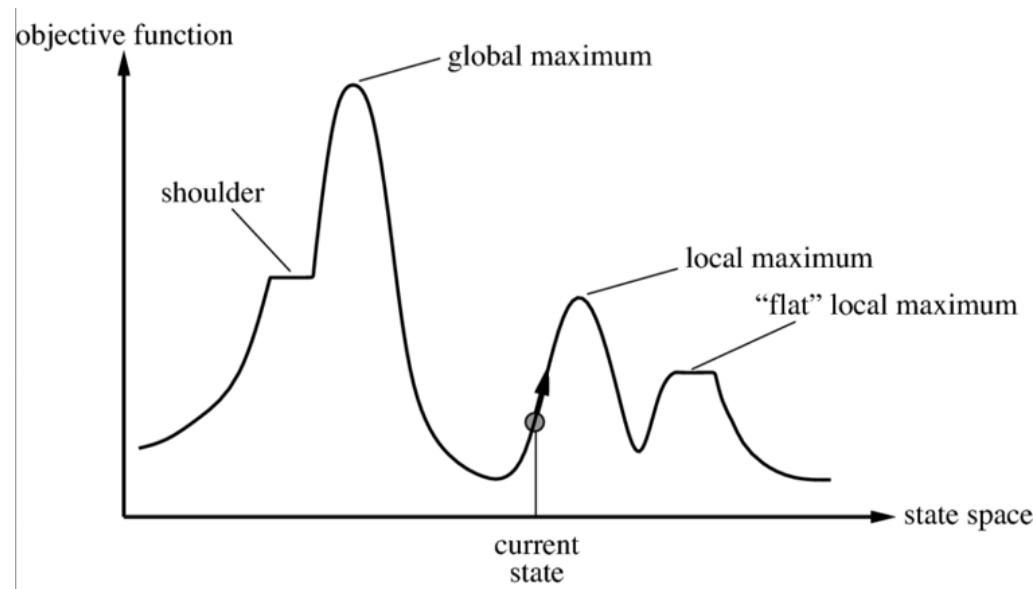


Recaptação

- Soluções aleatorizadas
 - Nos permitem balancear entre exploitation vs exploration
- Seria possível fazer uso de aleatoriedade como elemento de varredura no espaço de busca, mas ainda usar algum outro recurso para melhoria da solução encontrada por meio da aleatoriedade?

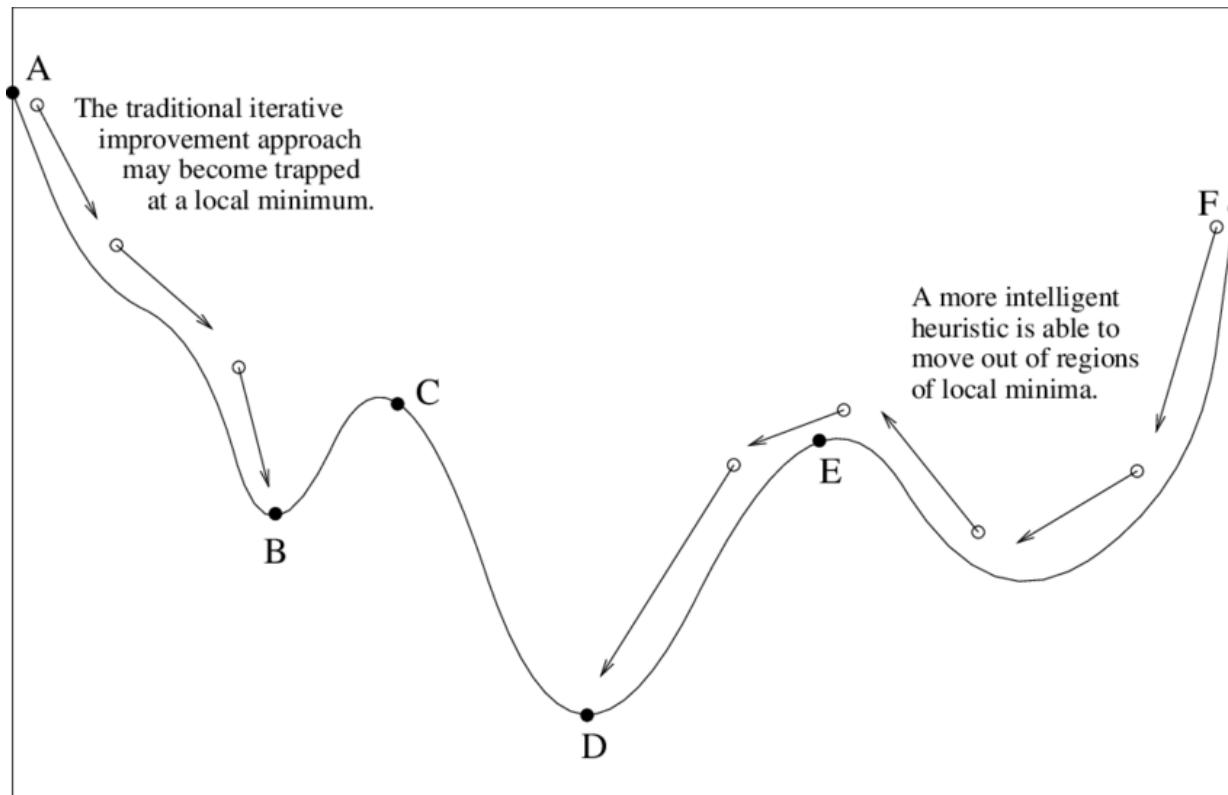
Problemas de otimização

- Em muitos problemas de otimização, o caminho para se atingir o objetivo é irrelevante, desde que a possamos conseguir uma solução para o problema em si



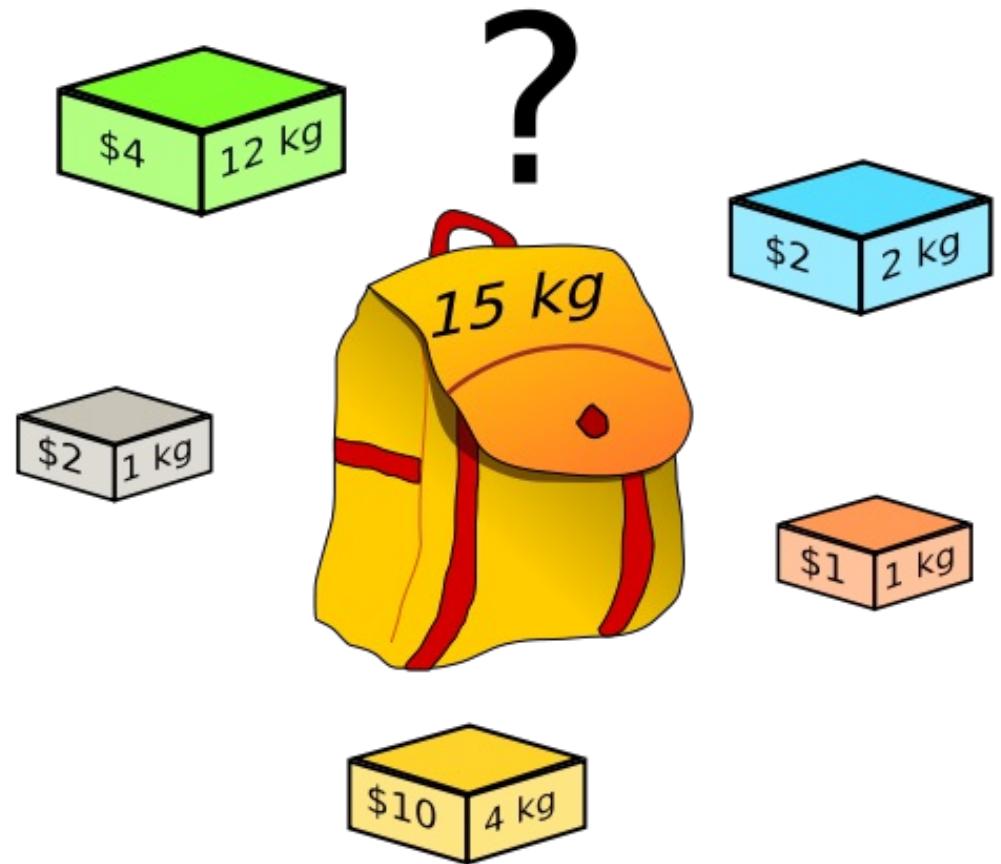
Melhorias após solução aleatória

- Exemplo:
minimização



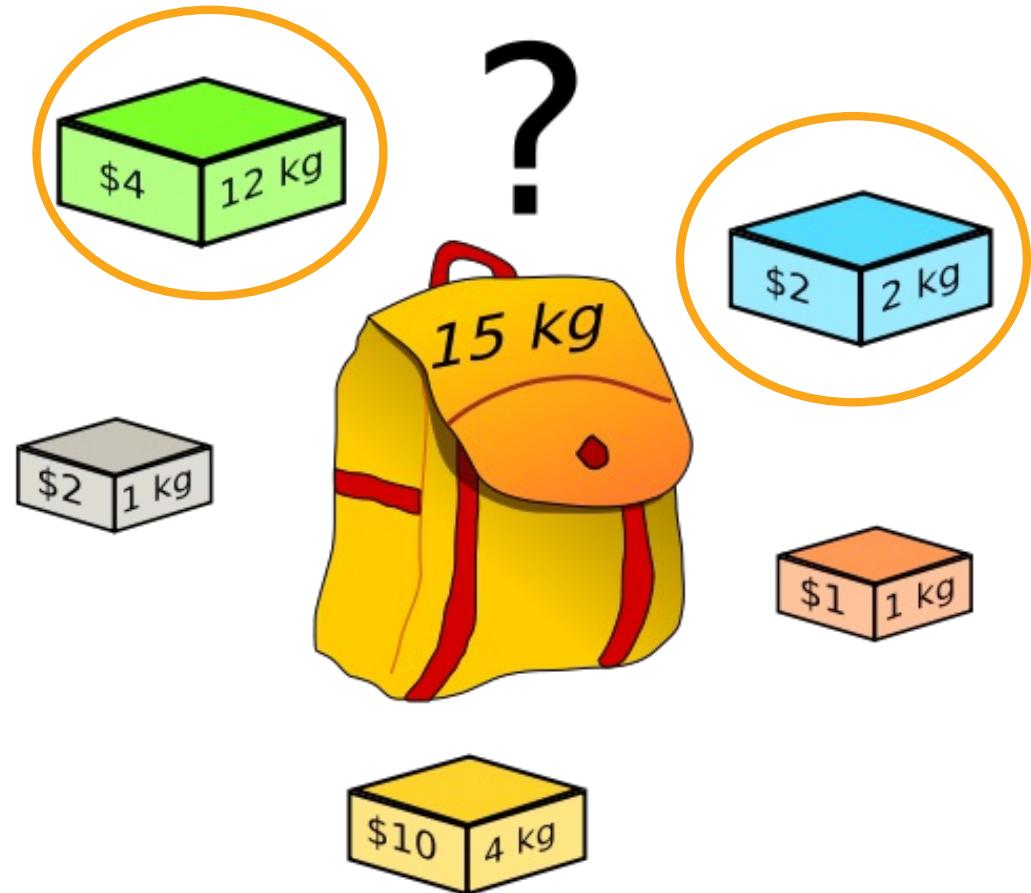
Mochila

- Uma possível solução aleatória:



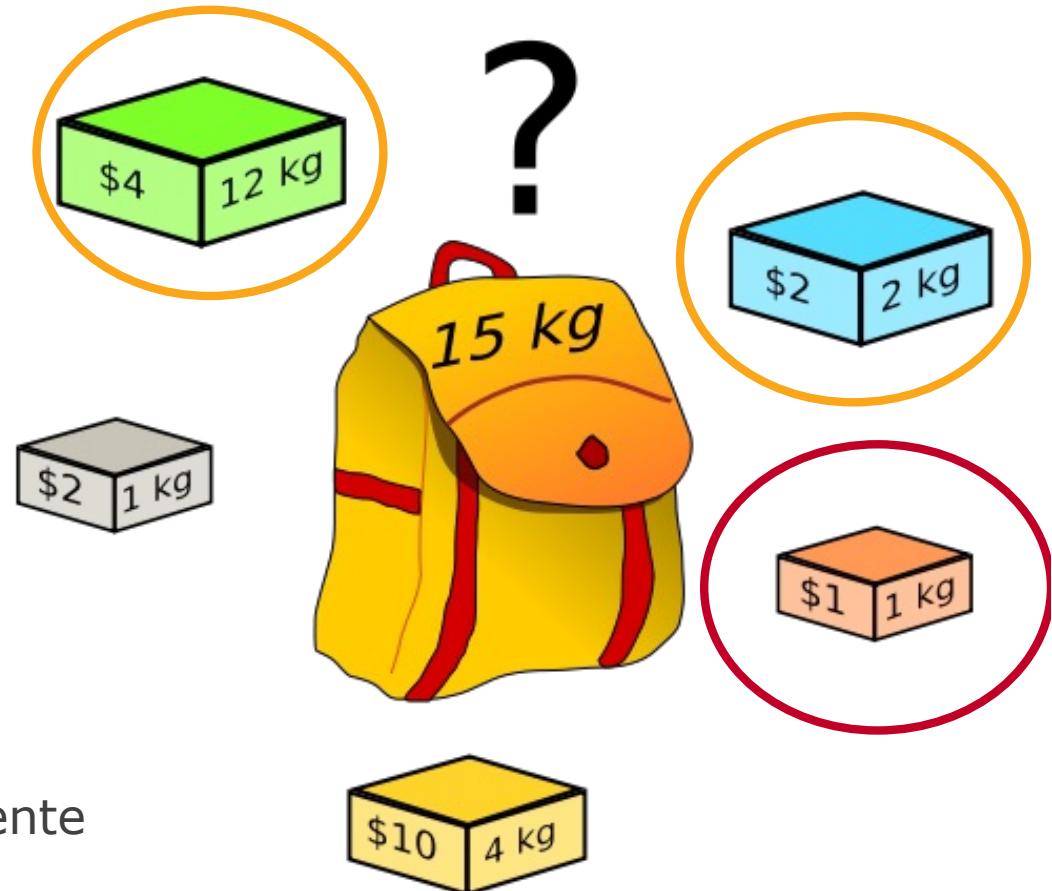
Mochila

- Uma possível solução aleatória:
 - Peso: 14 Kg
Valor: \$6



Mochila

- Uma possível solução aleatória:
 - Peso: 14 Kg
Valor: \$6
- Melhorando:
 - Adicionamos 1 item a mais
Peso: 15 Kg
Valor: \$7
- Ou seja, conseguimos melhorar uma solução gerada aleatoriamente



Como então podemos melhorar?

- Para a mochila, após gerarmos soluções iniciais aleatórias, podemos fazer duas ações:
 1. **Encher a mochila:** verificar se algum objeto não selecionado cabe na mochila
 2. **Trocar dois objetos:** verificar se é possível substituir um objeto selecionado por outro de melhor valor que foi deixado de fora

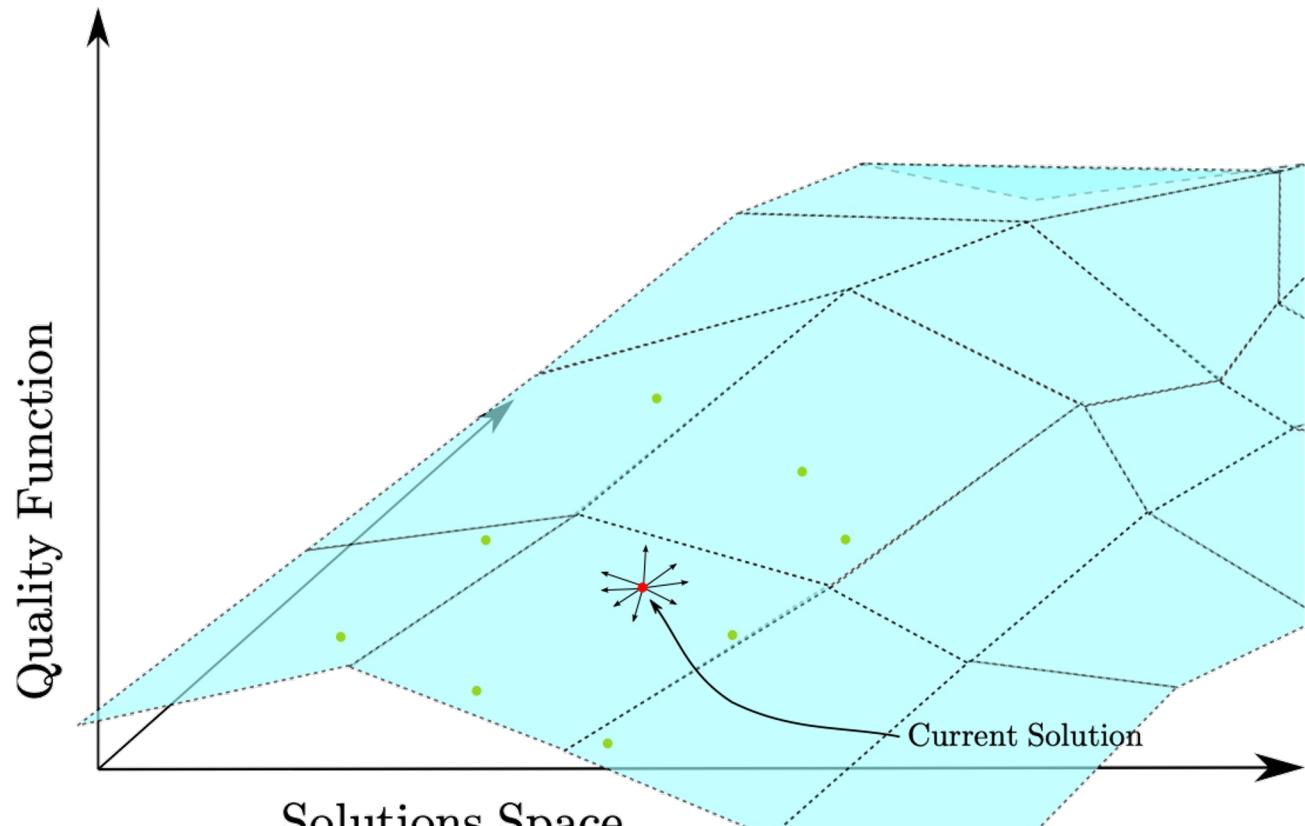
Como então podemos melhorar?

- Para a mochila, após gerarmos soluções iniciais aleatórias, podemos fazer duas ações:
 1. **Encher a mochila:** verificar se algum objeto não selecionado cabe na mochila
 2. **Trocar dois objetos:** verificar se é possível substituir um objeto selecionado por outro de melhor valor que foi deixado de fora
- Ambas são condições necessárias, mas não suficientes, para otimalidade
Ou seja, não há garantia de solução ótima global!**

Busca Local

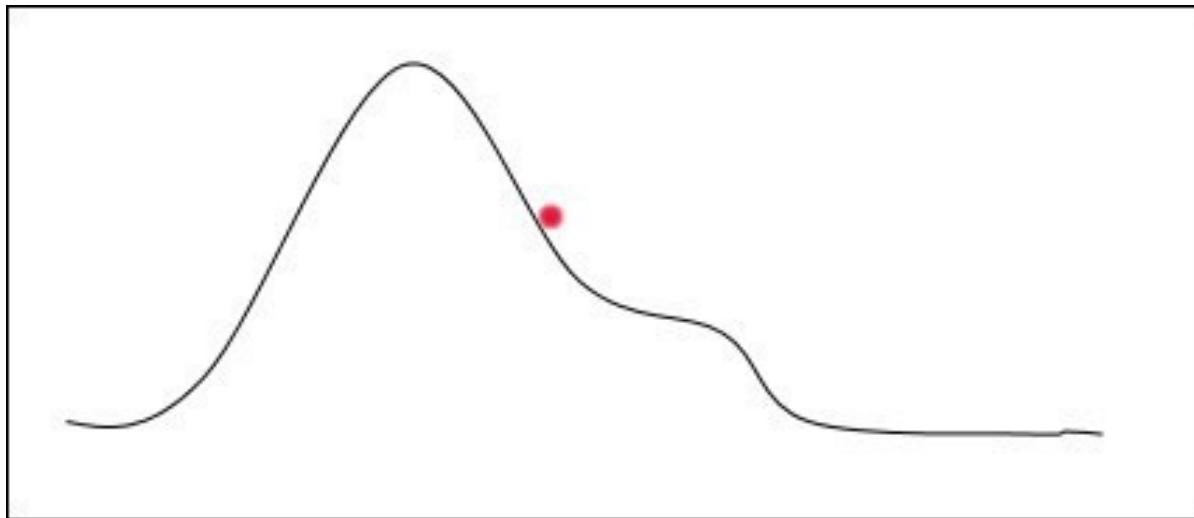
- Não precisamos do caminho para obter a solução, nós buscamos possíveis soluções por meio da aleatoriedade
- Vamos então obter uma solução inicial aleatória, em tempo plausível
- Temos que ter uma função que avalie a qualidade (fitness) dessa solução. Essa qualidade não está relacionada a trajetória da solução, apenas a solução em si
- A partir dessa solução inicial, fazemos uma busca na vizinhança de soluções (local), de modo a melhorar a qualidade da solução

Busca Local



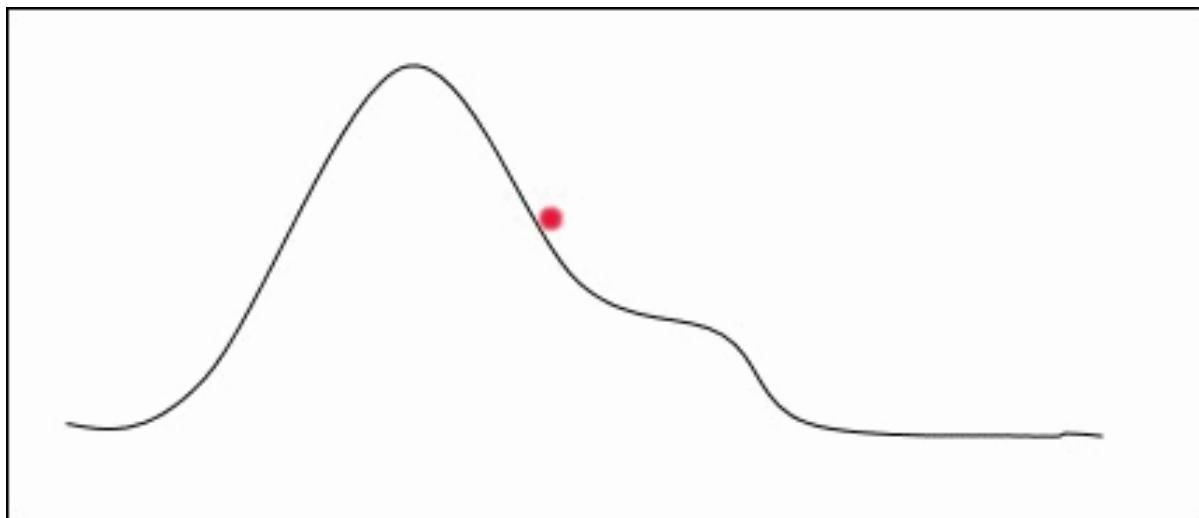
Busca Local – Hill Climbing

- Hill Climbing é um algoritmo clássico para otimização, bastante eficiente na tarefa de encontrar máximos ou mínimos locais
- Nós iniciamos em um ponto aleatório X e fazemos sua avaliação



Busca Local – Hill Climbing

- Nós movemos então do nosso ponto X para um novo ponto vizinho X'
 - Se esse ponto X' for uma solução melhor que X , ficamos nele e repetimos o processo
 - Caso contrário, voltamos para X e podemos visitar outro vizinho ou interromper



Hill Climbing para a Mochila

- Suponha uma mochila com capacidade C e diversos itens, com seus respectivos pesos (W) e valores (V)
- Nosso objetivo: maximizar $\sum V$, respeitando a restrição $W \leq C$
- Podemos codificar nosso problema como uma string binária. 0 significa que o item i não foi incluído, enquanto que 1 significa que i foi incluído
- Supondo 10 objetos, nossa string poderia ser: 0010010000
 - Para este exemplo, vamos gerar 10 possíveis vizinhos a partir da modificação de um bit: **1**010010000, 0**1**10010000, etc.
 - Vamos computar a qualidade desses 10 vizinhos. Se houver um melhor, ele é a nova solução e repetimos o processo, até que nenhum vizinho melhor seja encontrado

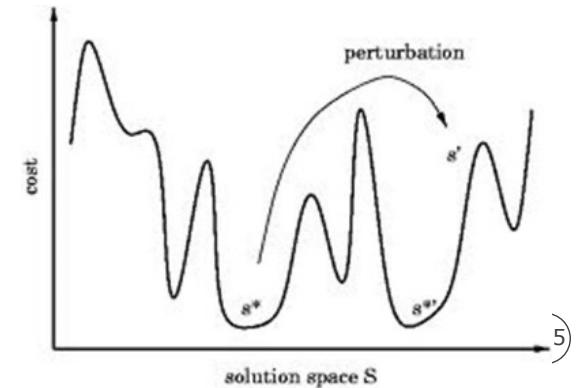
Busca local com perturbação

- Ideia baseada em dois estágios:
 - Gera uma solução inicial aleatória, e realiza busca local (hill climbing) [intensificação]
 - Faz uma perturbação na melhor solução encontrada, para evitar máximo local [diversificação]
- Desafio: controlar intensificação vs diversificação (critério de parada)

Algoritmo Busca local iterada (ILS):

INICIO

```
s = inicializa (s)
s = hill-climbing (s)
ENQUANTO NÃO critério_parada
    r = s
    s = perturbação (s)
    s = hill-climbing (s)
    SE s < r
        s = r
FIM
```

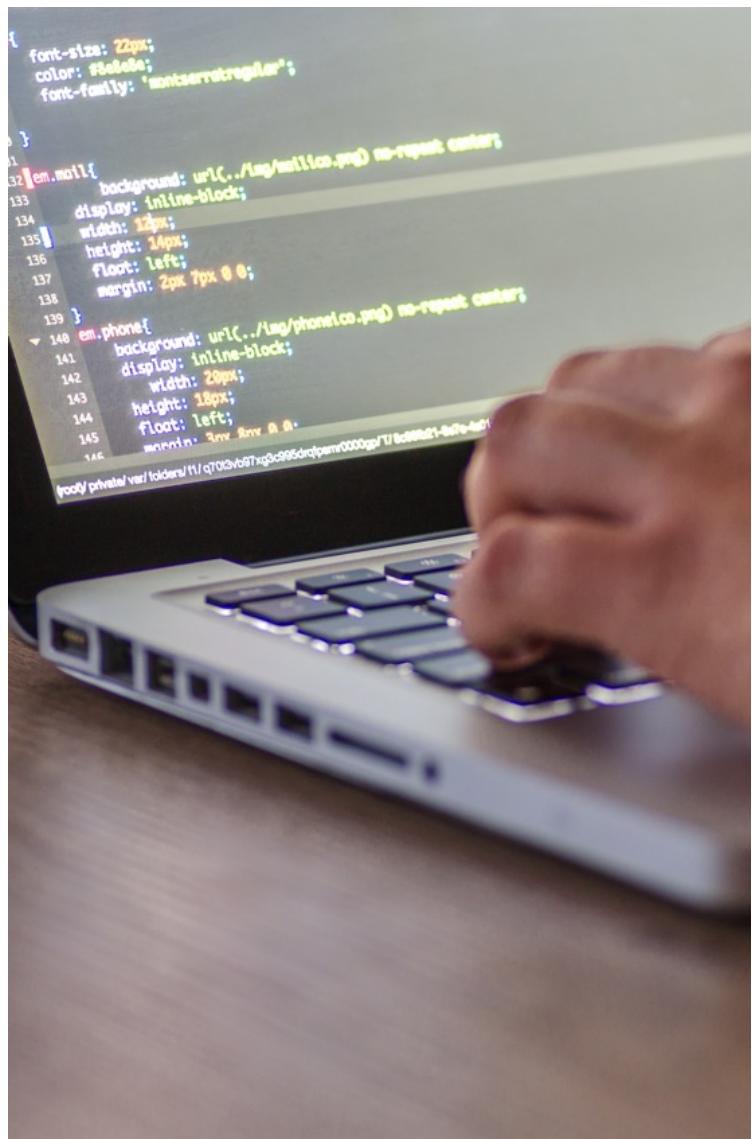


Busca local - vantagens

- Rápida
- Resultados bons para N grande
- Oferece garantia fraca de qualidade (máximos/mínimos locais)
- Não ficou bom? Rode mais vezes

Busca local - desvantagens

- Depende da solução inicial
- Aleatorizada (o que nem sempre é um problema)
- Oferece garantia fraca (máximos/mínimos locais) de qualidade
- Estratégia de perturbação pode ajudar



Atividade prática

Resolvendo a mochila binária por meio de busca local



Discussão

- As soluções ficaram melhores que as heurísticas?
- E o tempo de execução?



Obrigado



Insper Supercomputação

Aula - 08

- Busca Exaustiva



Busca exaustiva – Força bruta

- Força bruta é normalmente a primeira ideia para resolver problemas computacionais
- Mas se para uma mochila nós temos n possíveis itens, então temos 2^n possíveis combinações para serem testadas
- Para a mochila binária, a complexidade do algoritmo é $O(2^n)$, o que nos limita a executar esse approach apenas para pequenos valores de n



Busca exaustiva na mochila

- Imagine uma mochila com capacidade 16 e 4 itens:

<i>item</i>	<i>weight</i>	<i>value</i>
1	2	\$20
2	5	\$30
3	10	\$50
4	5	\$10



Busca exaustiva na mochila

- Possíveis combinações

<i>Subset</i>	<i>Total weight</i>	<i>Total value</i>	
\emptyset	0	\$0	
{1}	2	\$20	
{2}	5	\$30	
{3}	10	\$50	
{4}	5	\$10	
{1,2}	7	\$50	
{1,3}	12	\$70	
{1,4}	7	\$30	
{2,3}	15	\$80	
{2,4}	10	\$40	
{3,4}	15	\$60	
{1,2,3}	17	not feasible	
{1,2,4}	12	\$60	
{1,3,4}	17	not feasible	
{2,3,4}	20	not feasible	
{1,2,3,4}	22	not feasible	

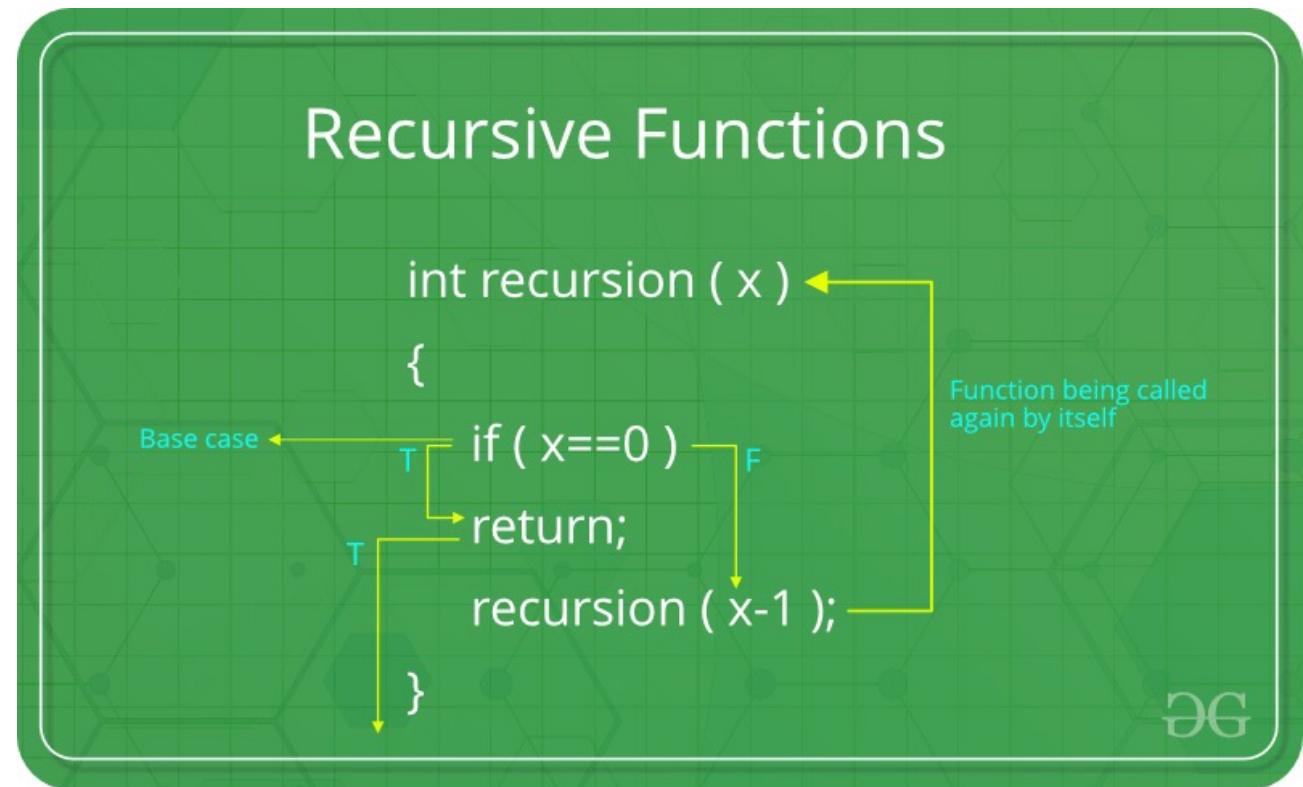
Obtendo uma solução ótima global

- Para todo objeto, só temos duas possibilidades:
- (1) Incluir na mochila
- (2) Não incluir na mochila

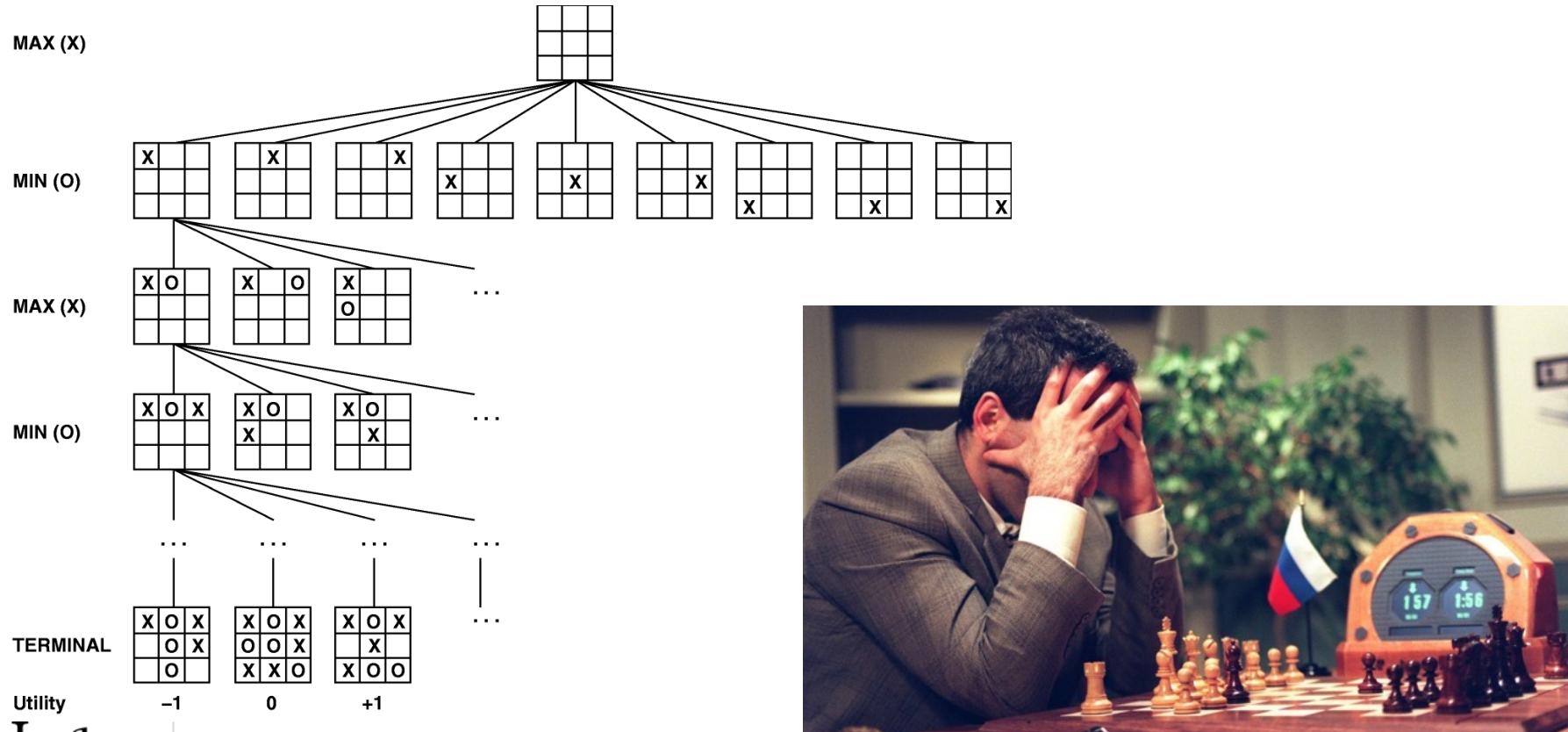
Obtendo uma solução ótima global

- Para todo objeto, só temos duas possibilidades:
- (1) Incluir na mochila
 - Resolva então agora um novo subproblema: uma mochila de menor capacidade (em função do novo item)
- (2) Não incluir na mochila
 - Resolva então agora um novo subproblema: a mochila tem a mesma capacidade, mas esse item foi removido da lista de itens

Esse algoritmo é **recursivo**



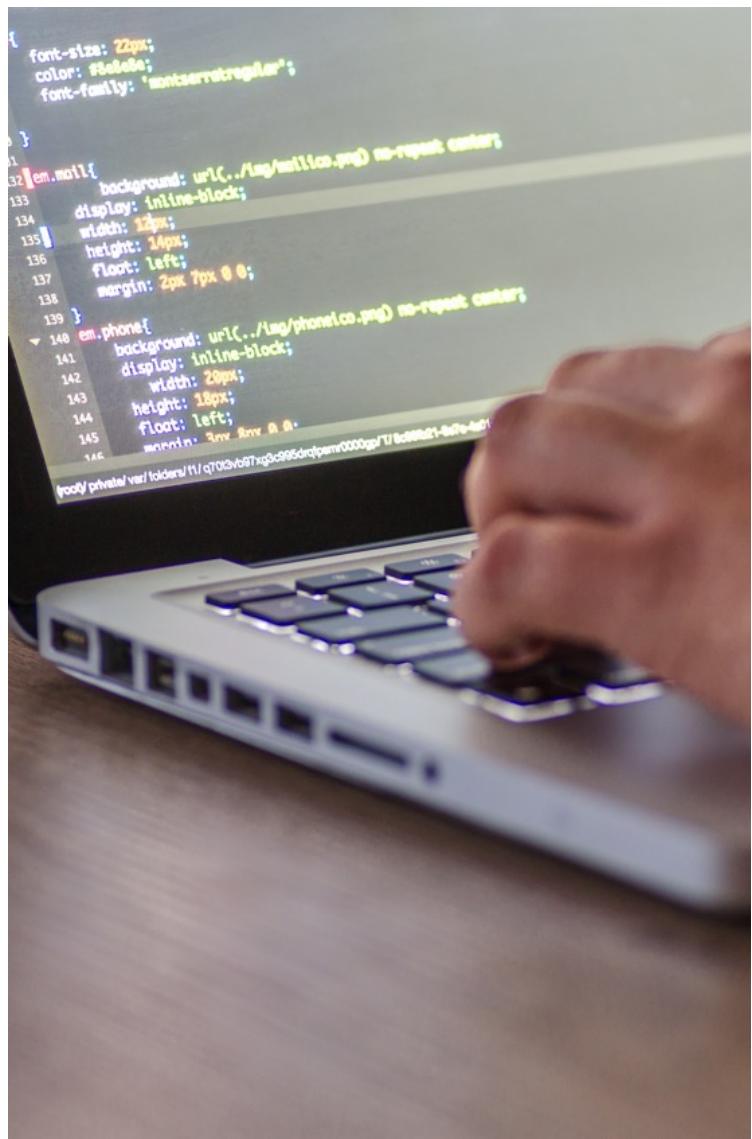
Busca exaustiva – inspiração MiniMax



It's time to have some fun



- **Vamos jogar Xadrez contra a máquina?**
- Vamos fazer isso a partir de uma versão do Minimax implementada em Java
- Avalie os seguintes pontos:
 - (i) é possível ganhar da máquina?
 - (ii) qual o tempo de processamento?



Atividade prática

Resolvendo a mochila binária de uma busca exaustiva recursiva

Pratique a implementação do algoritmo a partir do pseudocódigo disponível no roteiro da aula

Compare essa solução com outras abordagens (tempo, resultado, etc.)



Obrigado



Insper Supercomputação

Aula - 09

- Comparação de resultados

Aula 09

- Compilação de resultados – seguir roteiro



Obrigado



Insper Supercomputação

Aula - 10

- Acelerando a busca local
 - Branch and Bound

Branch and Bound

- Consiste em enumerar soluções viáveis, consideradas promissoras, em uma árvore de busca explícita ou implícita. Cada nó da árvore pode ser visto como uma possível solução ou uma solução parcial do problema

Branch and Bound

- Consiste em enumerar soluções viáveis, consideradas promissoras, em uma árvore de busca explícita ou implícita. Cada nó da árvore pode ser visto como uma possível solução ou uma solução parcial do problema
- Utilizando funções matemáticas e/ou algoritmos que determinam limites superiores (upper bound) e inferiores (lower bound) do problema com respeito ao valor ótimo da função objetivo, uma grande quantidade de nós da árvore podem ser descartados, reduzindo assim, não só o espaço de memória, mas também o tempo de execução do algoritmo

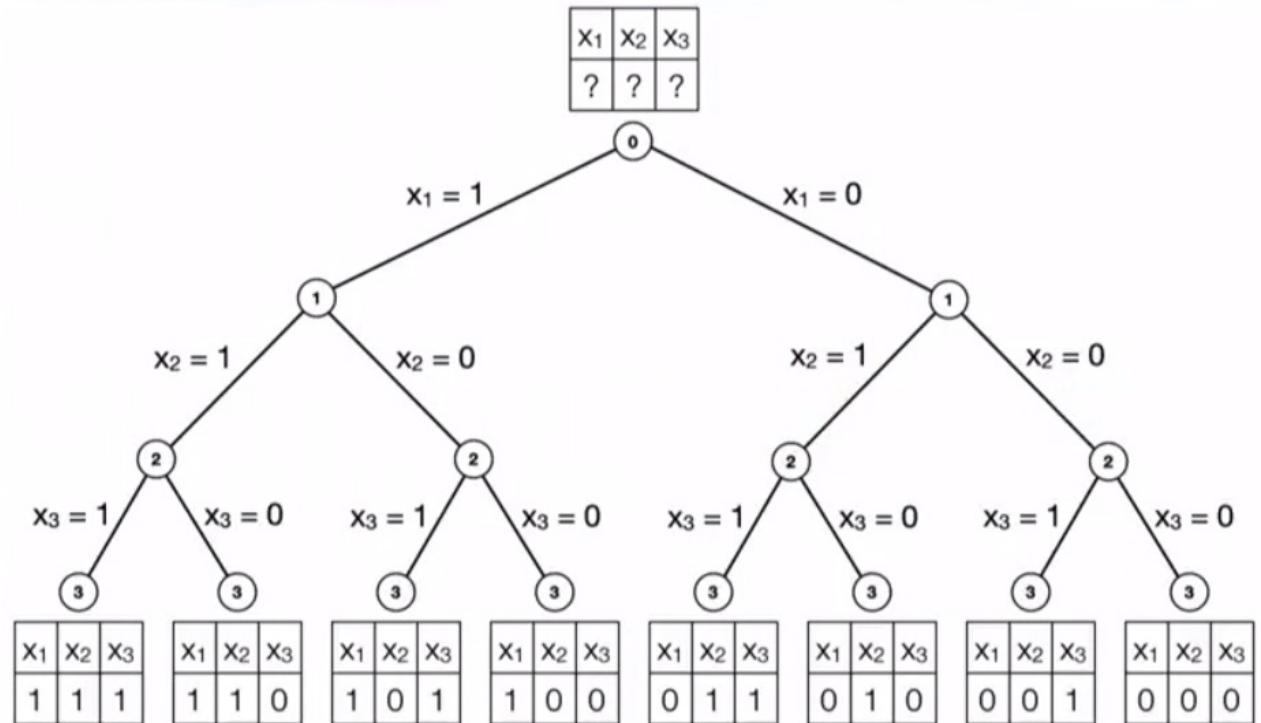
Branch and Bound

- Branch
 - Indica que a partir de um nó selecionado, deve ser possível explorar outros nós como alternativas de solução para o problema
- Bound
 - Sugere a identificação de dois limites com relação ao valor ótimo do problema: inferior e superior
 - O valor obtido pela solução ótima do problema, se existir, é encontrado em um intervalo delimitado por estes dois limites. O método vai reduzindo esse intervalo até que estes dois limites sejam iguais, quando, portanto, encontra-se o ótimo

Ideia

- Essa é a busca exaustiva
- Qual o papel de branch-and-bound?

Evitar terminar uma solução parcial que não tem chance alguma de ser ótimas



Ideia

- Nossa tarefa
 - Maximizar o valor da mochila
- Dois passos interativos: branching e bouding
- **Branching:** dividir o problema em subproblemas menores
- **Bounding:** encontrar uma **estimativa ótima** da melhor solução para cada subproblema
 - Upper bound a ser maximizado
 - Lower bound a ser minimizado
- Como determinar a estimativa ótima?
 - R: relaxamento de restrições do problema

Matematicamente

- Vamos enxergar o problema como uma programação linear inteira
- $C = 10$, 3 itens: $X_0 = 45/5\text{Kg}$, $X_1 = 48/8\text{Kg}$ e $X_2 = 35/3\text{Kg}$
- Logo:

$$\max 45X_0 + 48X_1 + 3X_2$$

Sujeito a

$$5X_0 + 8X_1 + 3X_2 \leq 10$$

$$X_i \in \{0,1\}$$

Matematicamente

- Vamos enxergar o problema como uma programação linear inteira
- C = 10, 3 itens: $X_0 = 45/5\text{Kg}$, $X_1 = 48/8\text{Kg}$ e $X_2 = 35/3\text{Kg}$
- Logo:

$$\max 45X_0 + 48X_1 + 35X_2$$

Sujeito a

$$\underline{5X_0 + 8X_1 + 3X_2 \leq 10}$$

$$X_i \in \{0,1\}$$

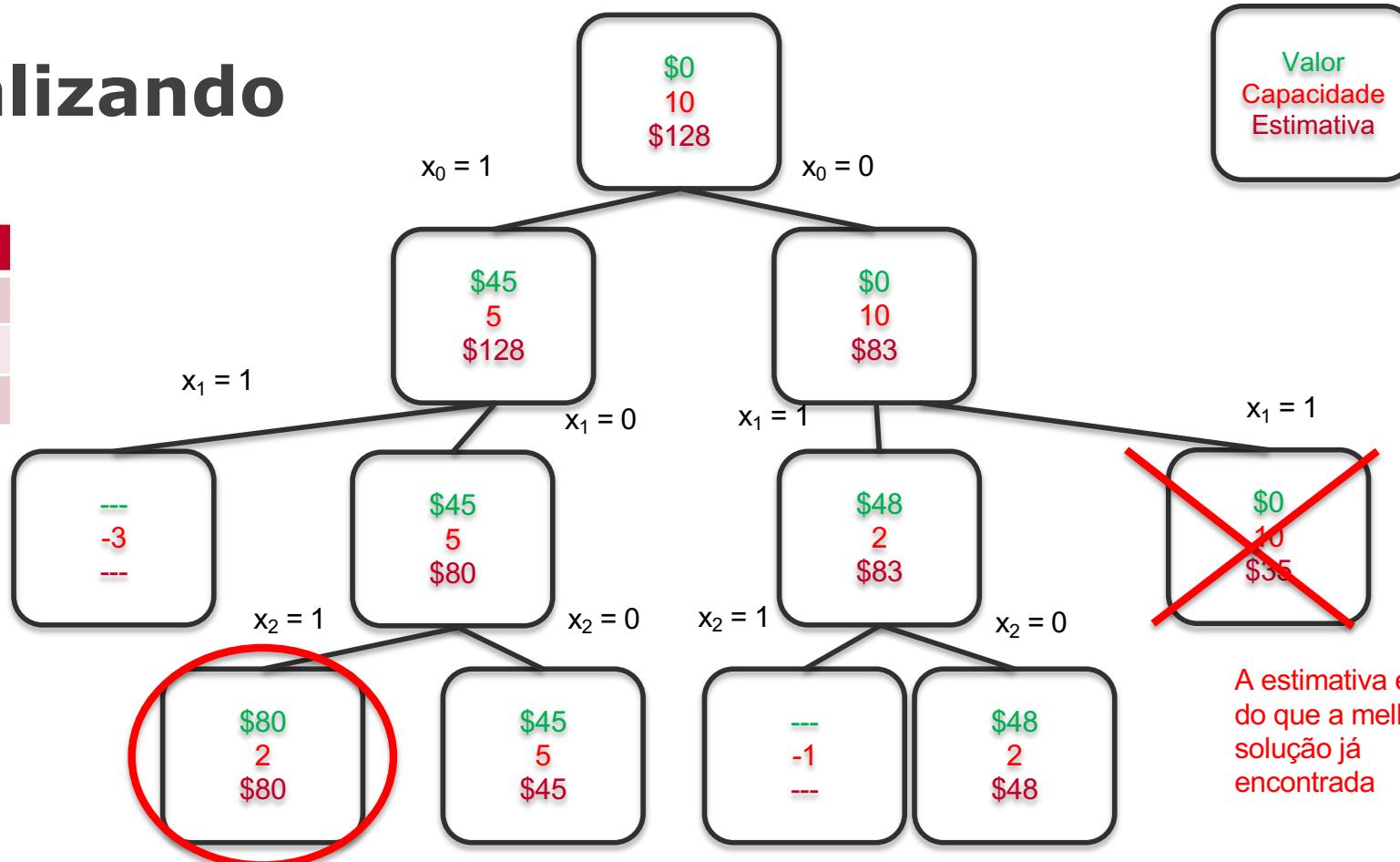
Vamos relaxar a constraint de capacidade máxima da mochila, para obter a estimativa de valor ótimo de um subproblema

Ou seja, assumimos que a capacidade estimada de valor de um subproblema é dada por seu valor atual mais a inclusão de todos os itens restantes, independente da capacidade da mochila

Visualizando

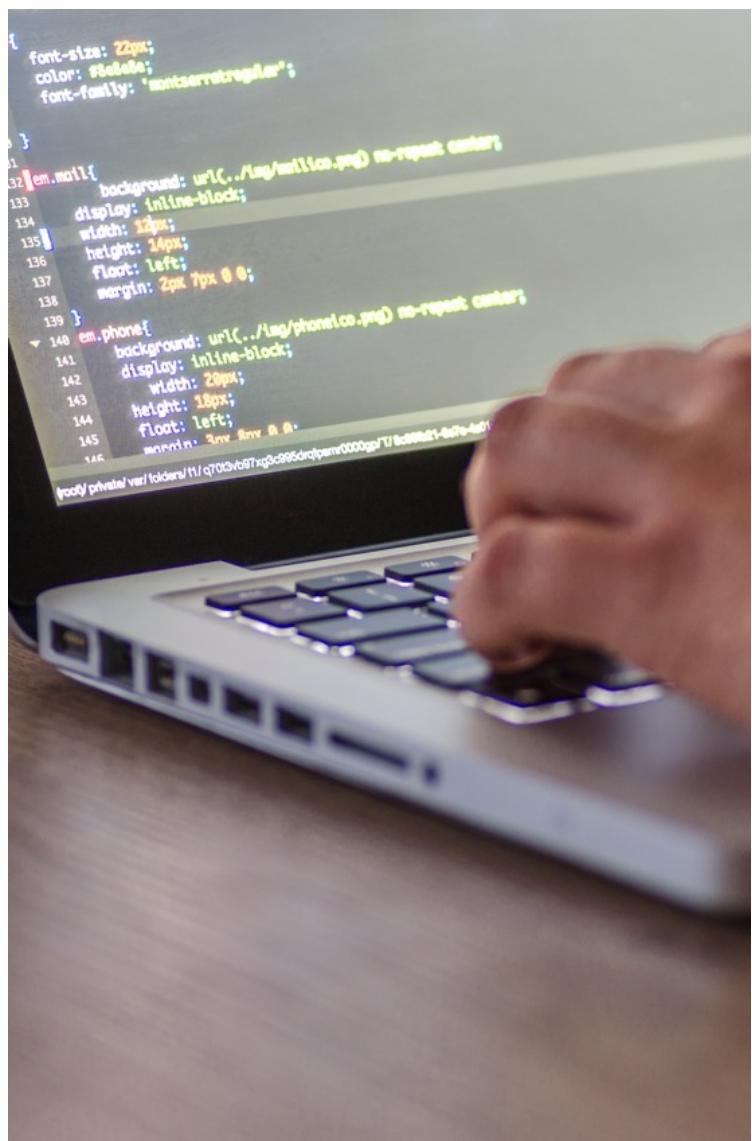
$S = 10$

i	v _i	s _i
0	45	5
1	48	8
2	35	3



Valor
Capacidade
Estimativa

A estimativa é pior
do que a melhor
solução já
encontrada



Atividade prática

Vamos implementar, seguindo roteiro da aula, a estratégia de Branch and bound para a mochila, com a estratégia de ignorar a capacidade da mochila

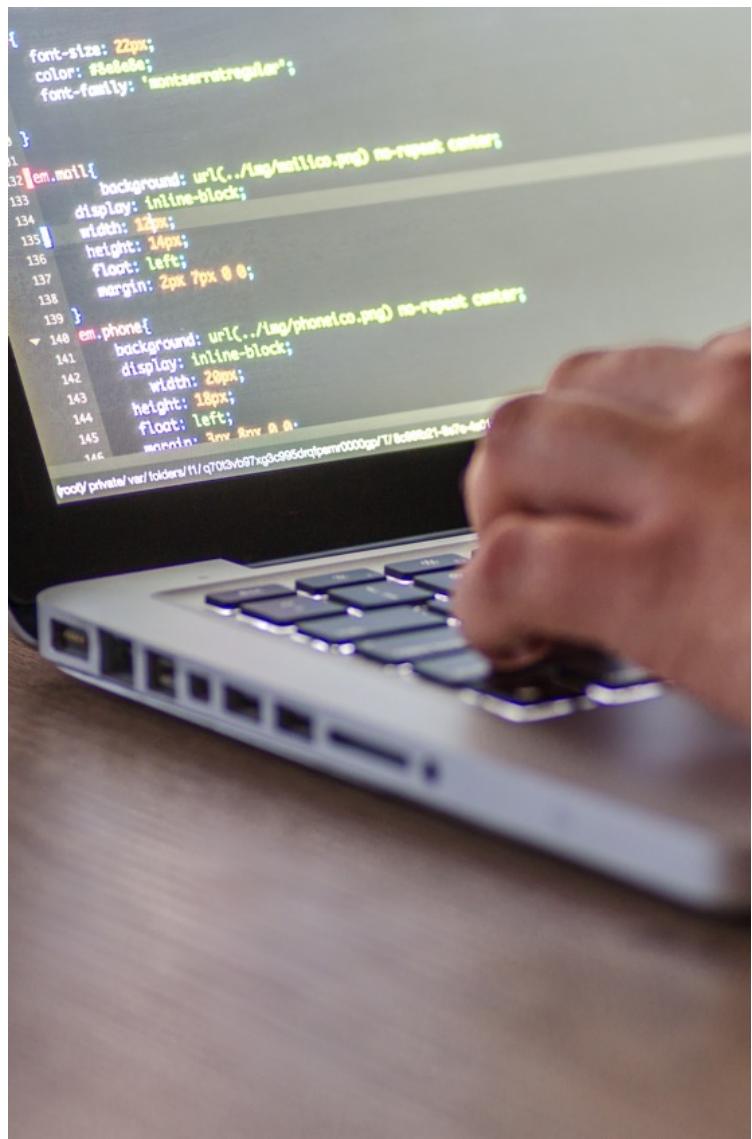
Você vai encontrar o pseudocódigo disponível no roteiro.

Compare a solução obtida com as versões anteriores.



Discussão

- Como descobrir se um bound é bom?
 - Quantas vezes ele é ativado?
 - Em qual altura ele é ativado?
 - O quanto bem ele estima a capacidade da solução parcial?



Atividade prática

Vamos implementar, de acordo com o roteiro, estratégias para compreender se o bound é bom.

Pense também em oportunidades para economizar o trabalho computacional



Obrigado



Insper Supercomputação



Aula - 11

- Paralelismo



Recaptulando...

- Soluções de alto desempenho
- (1) Algoritmos eficientes
- (2) Implementação eficiente (cache, por exemplo)
- (3) Paralelismo

Paralelismo

- Consiste no uso de múltiplos processadores, simultaneamente, para resolver um problema
- Tem por objetivo o aumento do desempenho, i.e., a redução do tempo necessário para resolver um problema

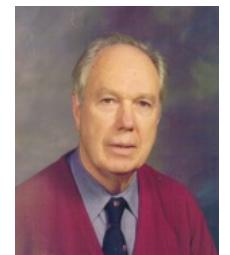


Paralelismo



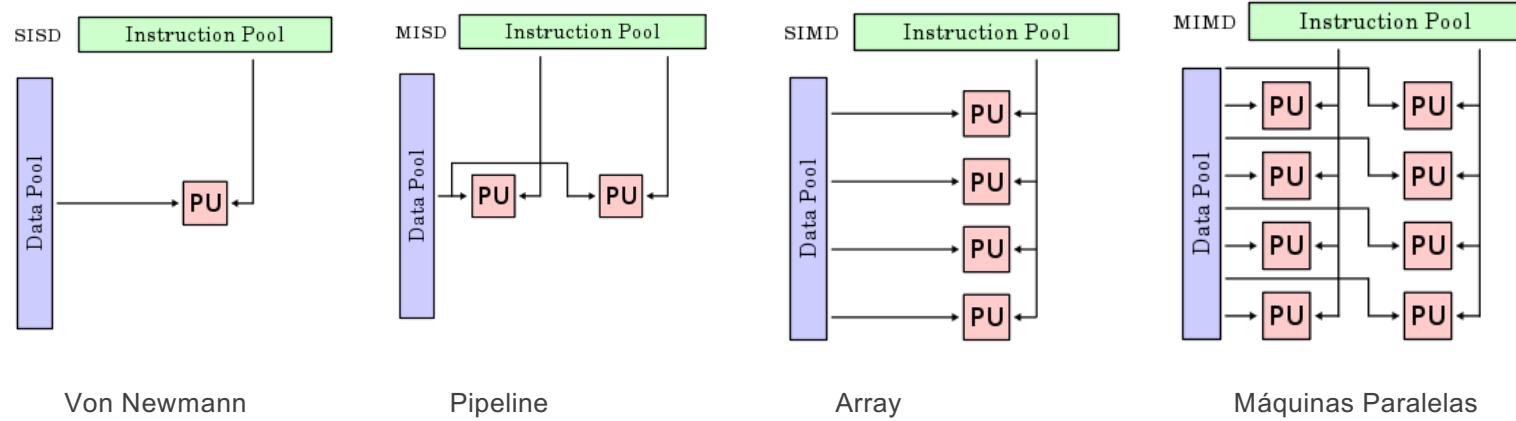
- Usamos paralelismo normalmente por 2 motivos
- (1) Problemas cada vez mais complexos e/ou maiores
- (2) Clock dos processadores se aproximando dos limites ditados pela física

Taxonomia de Flynn

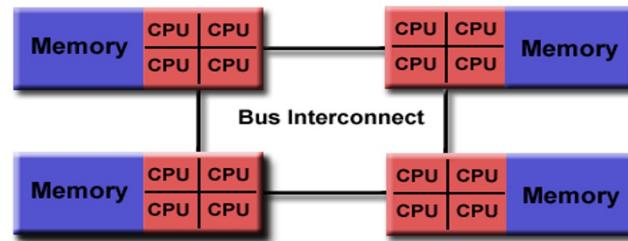
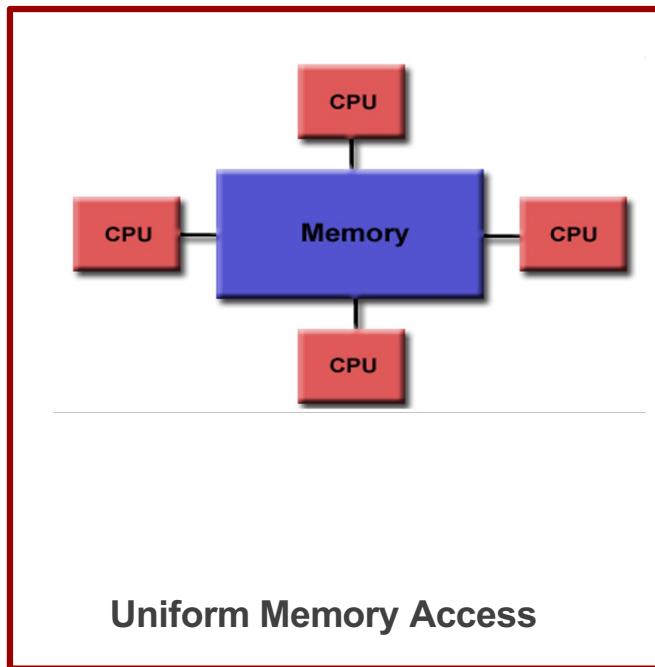


Michael J. Flynn

- É uma forma de classificar computadores paralelos
- Proposta por Flynn, em 1972
- Baseia-se no fato de um computador executar uma sequência de instruções sobre uma sequência de dados

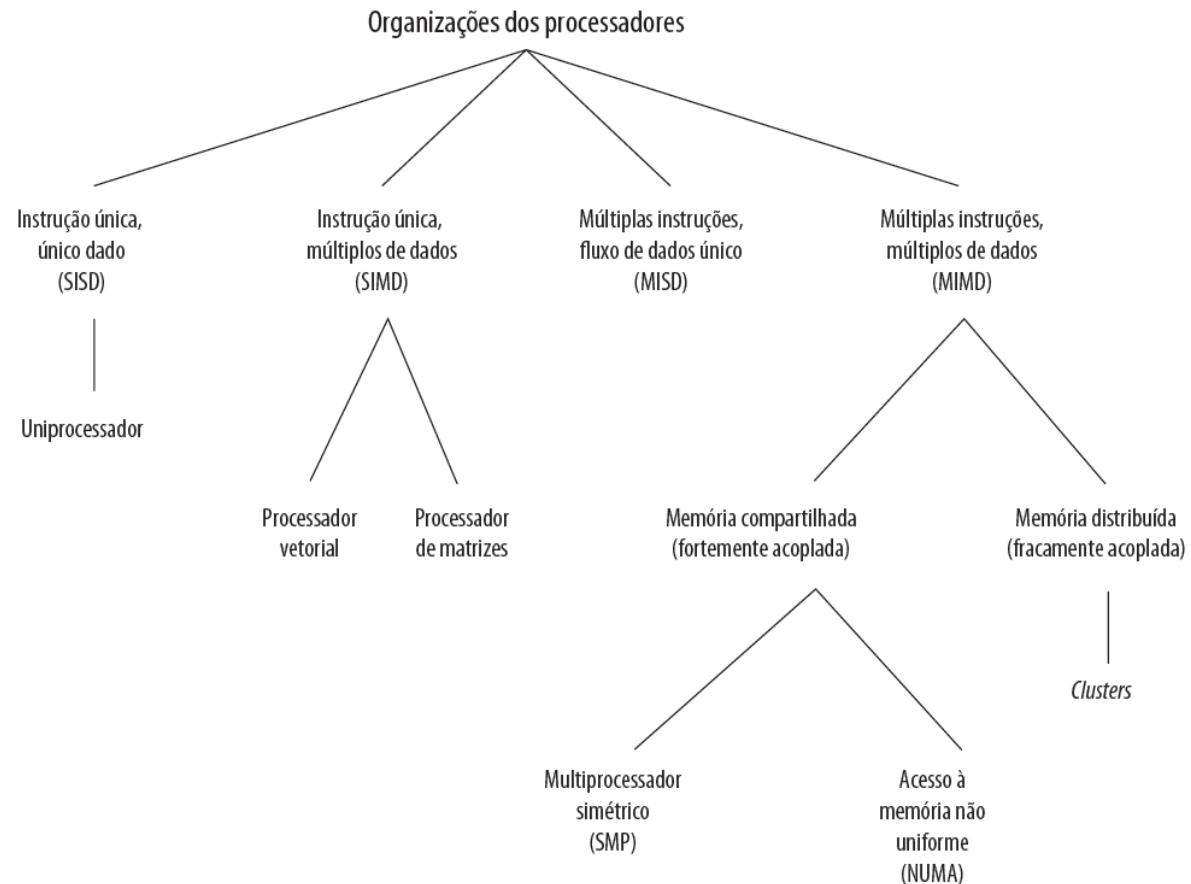


Sistemas Multi-core



Non-Uniform Memory Access

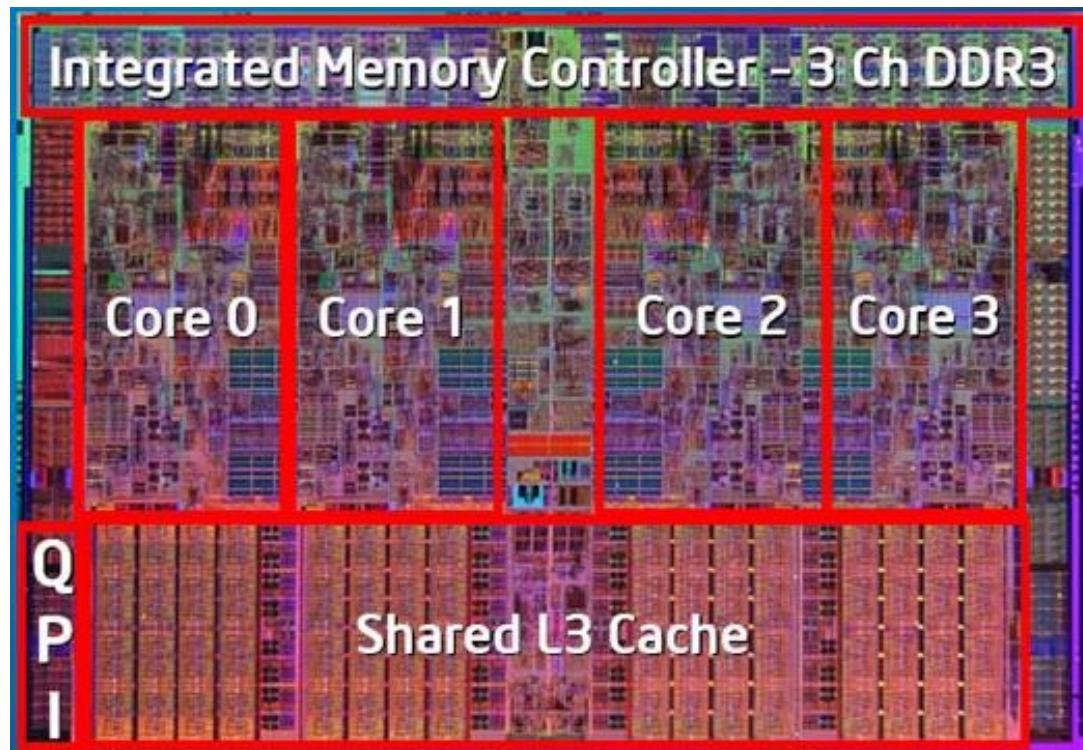
Organização dos processadores



Multiprocessadores simétricos

- Não faz muito tempo que todos os PCs continham um único processador de propósito geral
- À medida que a demanda por desempenho aumenta, e os custos de processadores são reduzidos, os fabricantes têm introduzido sistemas com uma organização SMP
- O termo SMP se refere a uma arquitetura de hardware computacional e também ao comportamento do sistema operacional que reflete essa arquitetura

Exemplo – Intel i7





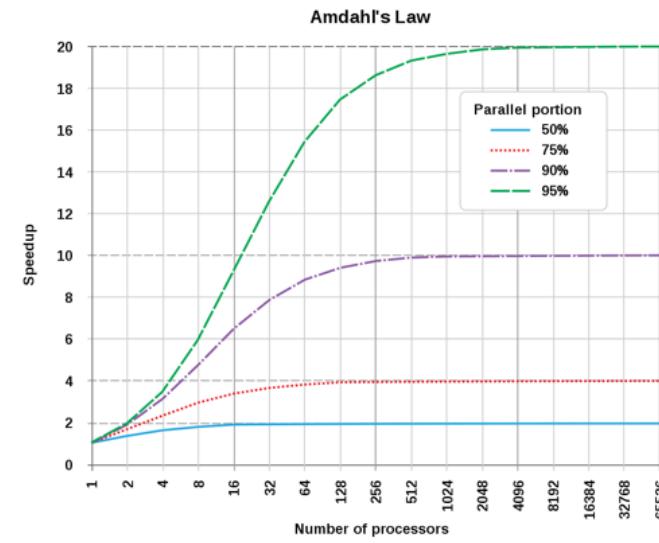
Discussão

- **Discussão 1: qual a expectativa da melhoria de velocidade com paralelismo?**



Discussão

- **Discussão 1: qual a expectativa da melhoria de velocidade com paralelismo?**

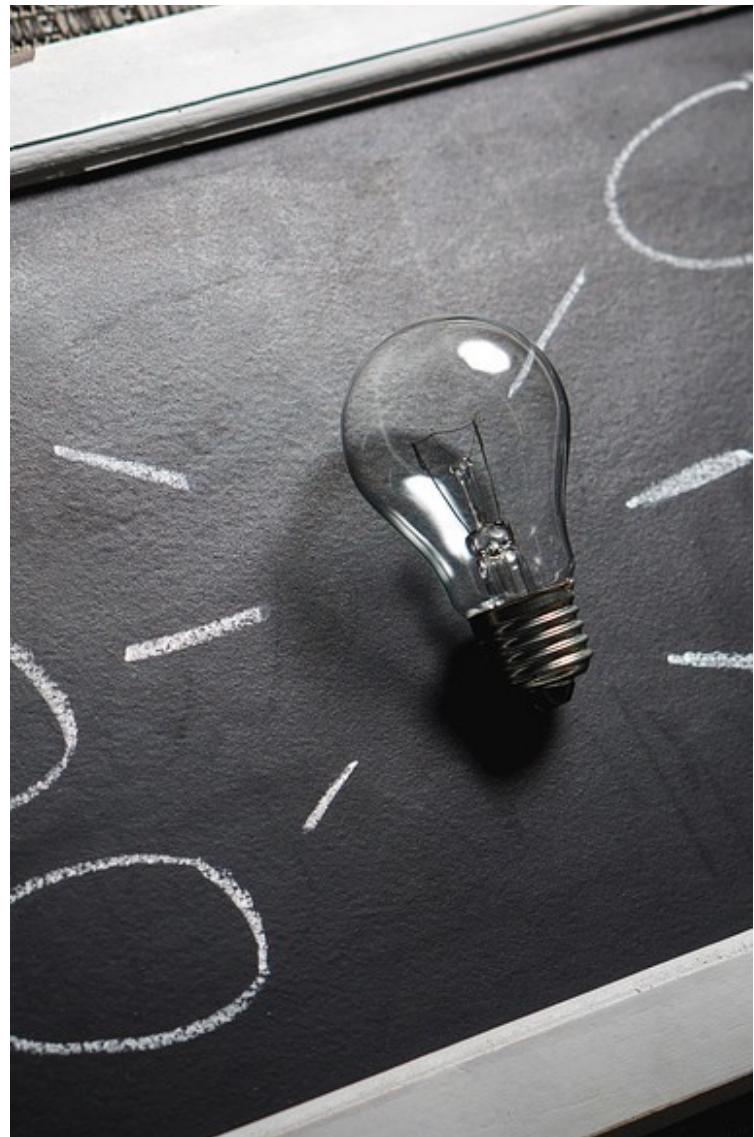




Discussão

- **Exemplo 1 – Supondo 8 cores**

```
vector<double> dados;
vector<double> resultados;
for (int i = 0; i < dados.size(); i++) {
    resultados[i] = funcao_complexa(dados[i]);
}
```



Discussão

- **Exemplo 1 – Supondo 8 cores**

```
vector<double> dados;
vector<double> resultados;
for (int i = 0; i < dados.size(); i++) {
    resultados[i] = funcao_complexa(dados[i]);
}
```

– Tempo total / 8



Discussão

- **Exemplo 2 – Supondo 8 cores**

```
vector<double> dados;
vector<double> resultados;
resultados[0] = 0;
for (int i = 1; i < dados.size(); i++) {
    resultados[i] = funcao_complexa(dados[i], resultados[i-1]);
}
```



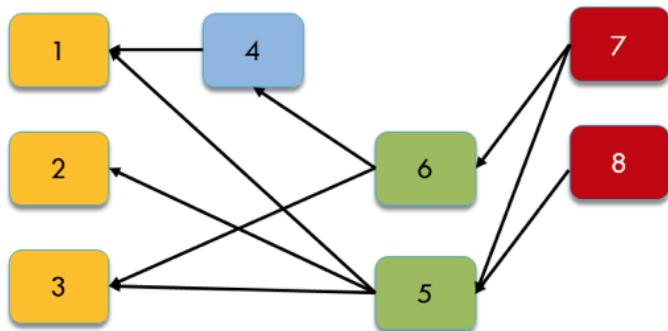
Discussão

- **Exemplo 2 – Supondo 8 cores**

```
vector<double> dados;
vector<double> resultados;
resultados[0] = 0;
for (int i = 1; i < dados.size(); i++) {
    resultados[i] = funcao_complexa(dados[i], resultados[i-1]);
}
```

Nenhum ganho! Depende da iteração anterior :(

Dependência



- É quando uma iteração depende de resultados calculados em iterações anteriores
- Quando não existe nenhuma dependência em um loop, por exemplo, dizemos que ele é ingenuamente paralelizável



Discussão

- **Exemplo 3 – Supondo 8 cores**

```
vector<double> dados;
vector<double> resultados1;
vector<double> resultados2;
resultados1[0] = resultados2[0] 0;
for (int i = 1; i < dados.size(); i++) {
    resultados1[i] = funcao_complexa(dados[i], resultados1[i-1]);
    resultados2[i] = funcao_complexa2(dados[i], resultados2[i-1]);
}
```

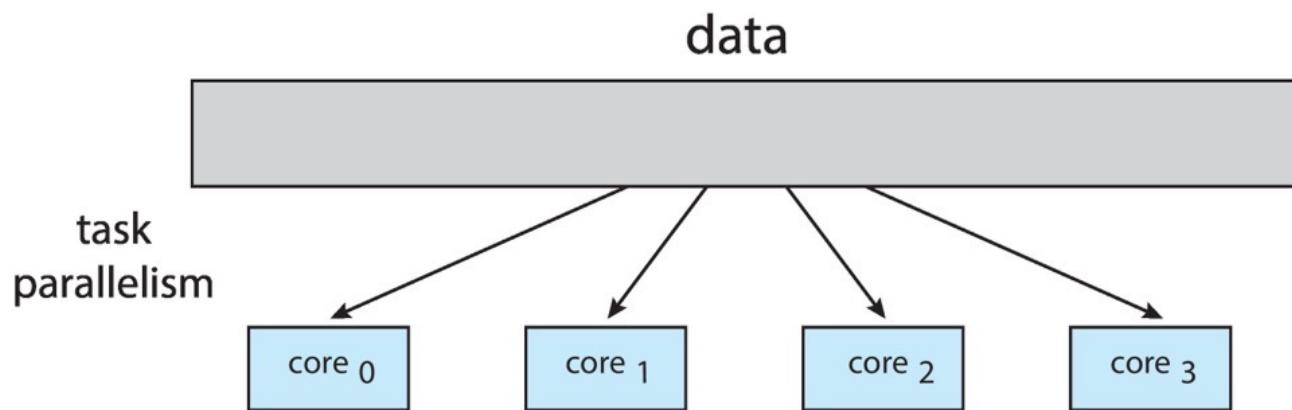
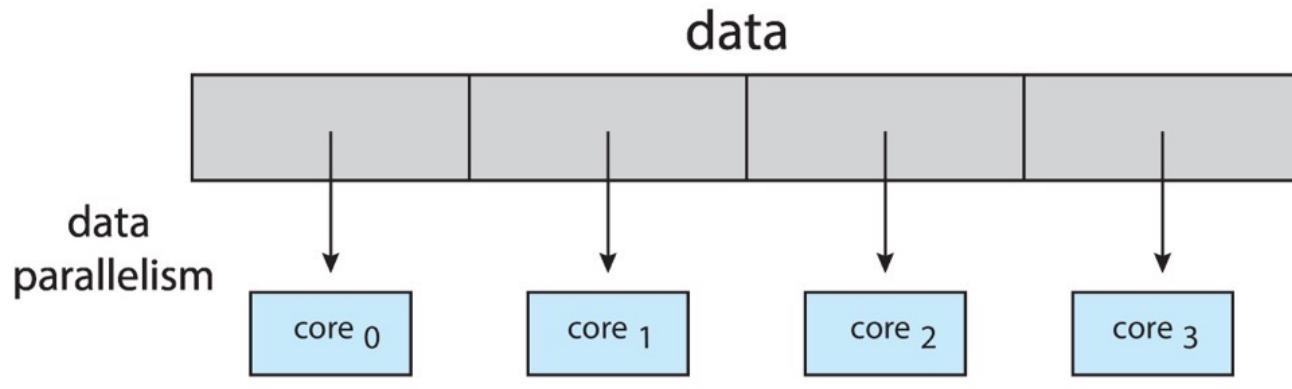
Podemos calcular resultados1 e resultados2 paralelamente

Paralelismo

- Paralelismo de dados: a mesma operação (lenta) é executada para todos os elementos de um conjunto de dados (grande)
- Paralelismo de tarefas: duas ou mais tarefas independentes são executadas em paralelo. Se houver dependências, quebramos o problema em partes independentes e rodamos na ordem adequada



Paralelismo



```
def __init__(self, path=None, debug=False):
    self.file = None
    self.fingerprints = set()
    self.logduplicates = True
    self.debug = debug
    self.logger = logging.getLogger(__name__)
    if path:
        self.file = open(os.path.join(path, 'fingerprint.log'), 'a')
        self.file.seek(0)
        self.fingerprints.update(self._load_fingerprints())
    else:
        self._load_fingerprints()

    @classmethod
    def from_settings(cls, settings):
        debug = settings.getbool("DRAFT_DEBUG")
        return cls(job_dir(settings))

    def request_seen(self, request):
        fp = self.request_fingerprint(request)
        if fp in self.fingerprints:
            return True
        self.fingerprints.add(fp)
        if self.file:
            self.file.write(fp + os.linesep)

    def request_fingerprint(self, request):
        return request_fingerprint(request)
```

Paralelismo

- Vamos entender essa ideia de dependência executando códigos em Python por meio da biblioteca **Dask**, neste [link](#)

Paralelismo - Resumo

1. Paralelizar significa rodar código sem dependências simultaneamente
2. Paralelismo de dados: mesma tarefas, dados diferentes
3. Paralelismo de tarefas: heterogêneo
4. Existem tarefas inherentemente sequenciais
5. Ganhos são limitados a partes do programa

OpenMP

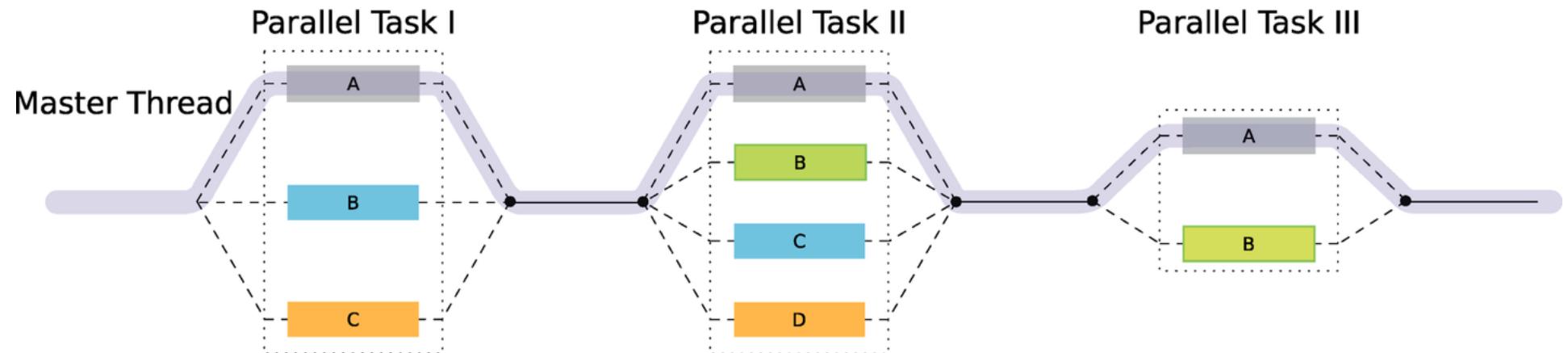
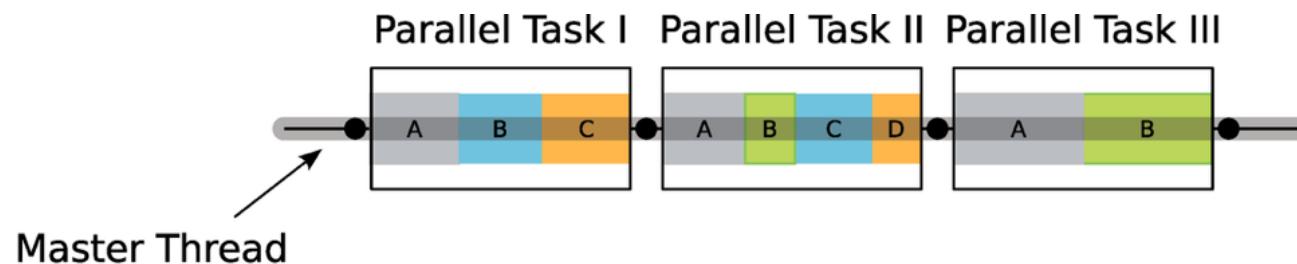
Sintaxe, Principais
Conceitos



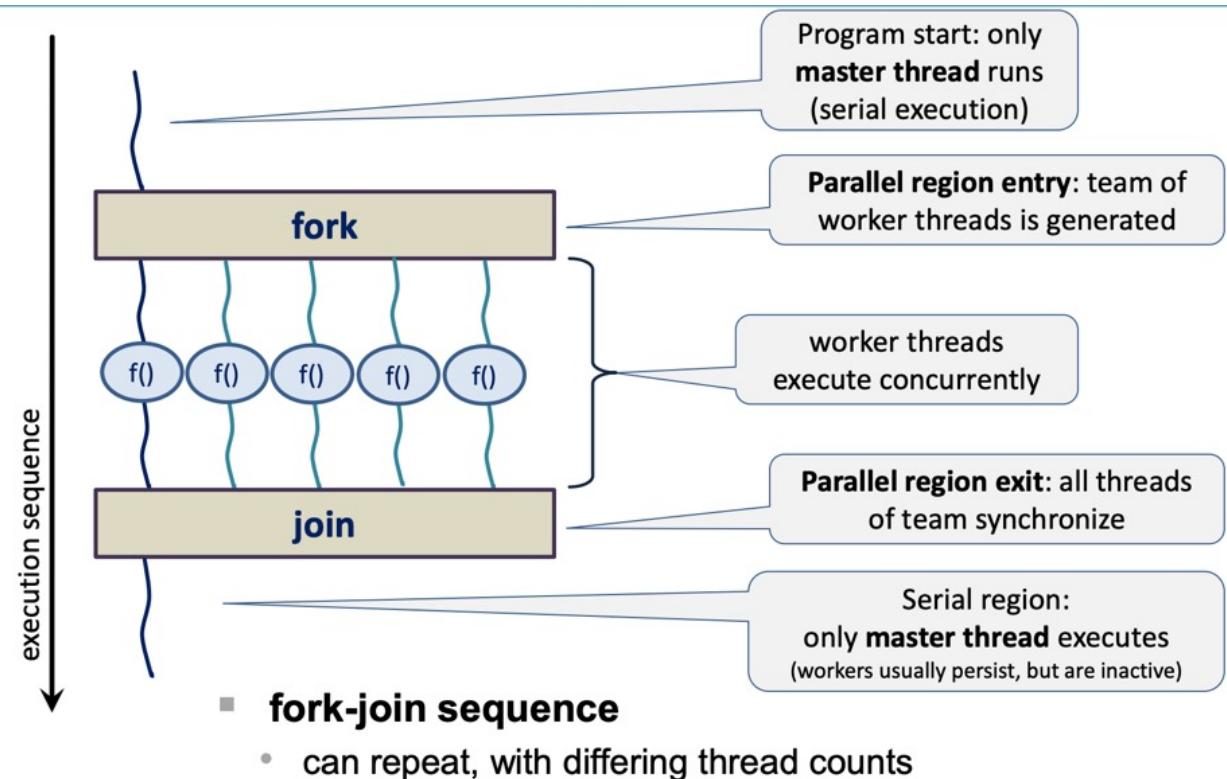
OpenMP

- Conjunto de extensões para C/C++ e Fortran
- Fornece construções que permitem paralelizar código em ambientes multi-core
- Padroniza práticas SMP + SIMD + sistemas heterogêneos (GPU/FPGA)
- Idealmente funciona com o mínimo de modificações no código sequêncial

OpenMP



OpenMP – Fork/Join



OpenMP – onde aprender mais

A brief Introduction to parallel programming

Tim Mattson
Intel Corp.
timothy.g.mattson@intel.com



Vídeos:

<https://www.youtube.com/watch?v=pRtTIW9-Nr0>
<https://www.youtube.com/watch?v=LRsQHDAqPHA>
<https://www.youtube.com/watch?v=dK4PITrQtjY>
https://www.youtube.com/watch?v=WvoMpG_QvBU

Slides:

http://extremecomputingtraining.anl.gov/files/2016/08/Mattson_830aug3_HandsOnIntro.pdf

OpenMP - compilação

To compile this with `gcc` we need to include the `-fopenmp` option:¹

```
$ gcc -g -Wall -fopenmp -o omp_hello omp_hello.c
```

OpenMP - Sintaxe

Diretivas de compilação

```
#include <omp.h>
#pragma omp construct [params]
```

Aplicadas a um bloco de código

limitado diretamente por { }

```
for (...) { }
```

Com join implícito

OpenMP - Sintaxe

```
// Arquivo interface da biblioteca OpenMP para C/C++
#include <omp.h>

// retorna o identificador da thread.
int omp_get_thread_num();

// indica o número de threads a executar na região paralela.
void omp_set_num_threads(int num_threads);

// retorna o número de threads que estão executando no momento.
int omp_get_num_threads();
```

OpenMP - Sintaxe

Name	Result type	Purpose
<code>omp_set_num_threads (int num_threads)</code>	none	number of threads to be created for subsequent parallel region
<code>omp_get_num_threads()</code>	int	number of threads in currently executing region
<code>omp_get_max_threads()</code>	int	maximum number of threads that can be created for a subsequent parallel region
<code>omp_get_thread_num()</code>	int	thread number of calling thread (zero based) in currently executing region
<code>omp_get_num_procs()</code>	int	number of processors available
<code>omp_get_wtime()</code>	double	return wall clock time in seconds since some (fixed) time in the past
<code>omp_get_wtick()</code>	double	resolution of timer in seconds

OpenMP - Sintaxe

```
// Cria a região paralela. Define variáveis privadas e
compartilhadas entre as threads.
#pragma omp parallel private(...) shared(...)
{ // Obrigatoriamente na linha de baixo.

// Apenas a thread mais rápida executa.
#pragma omp single

}
```

OpenMP - Sintaxe

Código sequencial

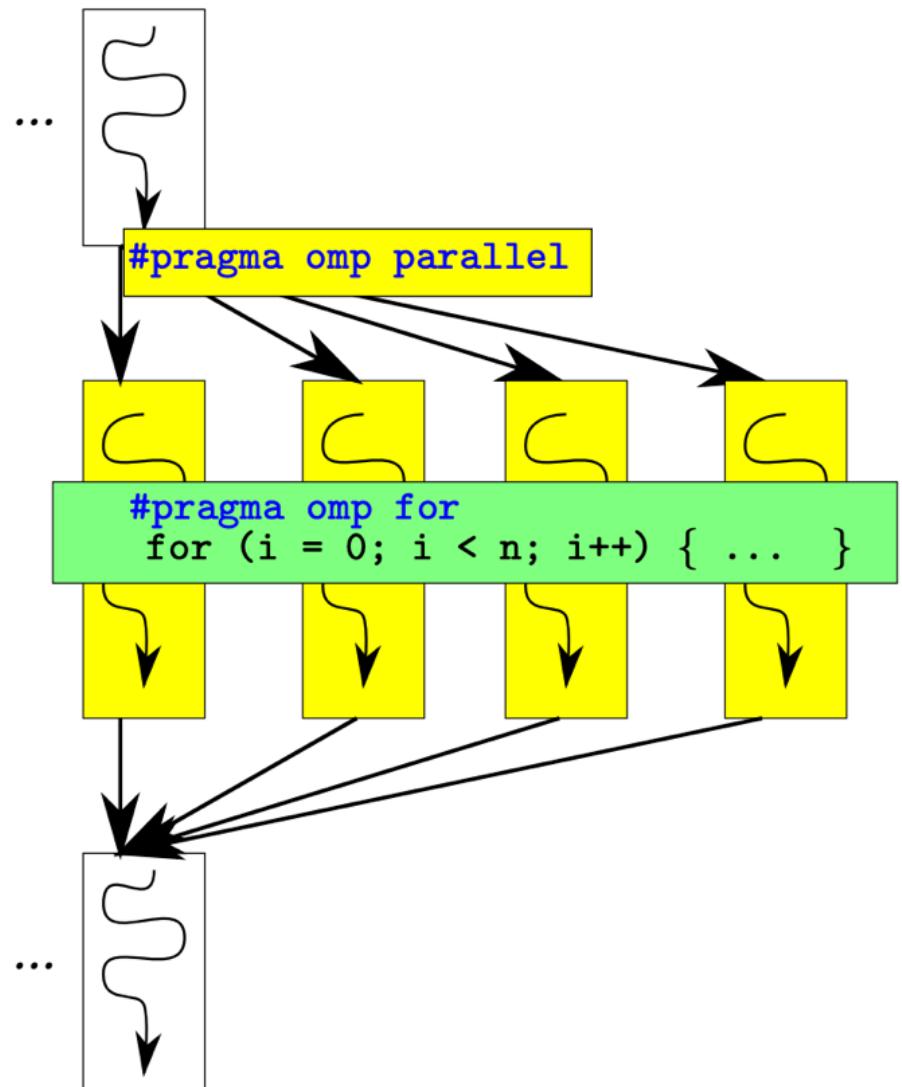
```
for(i = 0; i < N; i++)
    a[i] = a[i] + b[i];
```

Região OpenMP parallel

```
#pragma omp parallel
{
    int id, i, Nthrds, istart, iend;
    id = omp_get_thread_num();
    Nthrds = omp_get_num_threads();
    istart = id * N / Nthrds;
    iend = (id+1) * N / Nthrds;
    if(id == Nthrds-1) iend = N;
    for(i = istart; i < iend; i++)
        a[i] = a[i] + b[i];
}
```

OpenMP - Sintaxe

```
#pragma omp parallel
{
    int id, i, Nthrds, istart, iend;
    id = omp_get_thread_num();
    Nthrds = omp_get_num_threads();
    istart = id * N / Nthrds;
    iend = (id+1) * N / Nthrds;
    if(id == Nthrds-1) iend = N;
    for(i = istart; i < iend; i++)
        a[i] = a[i] + b[i];
}
```



OpenMP - Sintaxe

Código sequencial

```
for(i = 0; i < N; i++)
    a[i] = a[i] + b[i];
```

Região OpenMP parallel

```
#pragma omp parallel
{
    int id, i, Nthrds, istart, iend;
    id = omp_get_thread_num();
    Nthrds = omp_get_num_threads();
    istart = id * N / Nthrds;
    iend = (id+1) * N / Nthrds;
    if(id == Nthrds-1) iend = N;
    for(i = istart; i < iend; i++)
        a[i] = a[i] + b[i];
}
```

Região paralela OpenMP
com uma construção de
divisão de laço

```
#pragma omp parallel
#pragma omp for
for(i = 0; i < N; i++) a[i] = a[i] + b[i];
```

OpenMP - variáveis

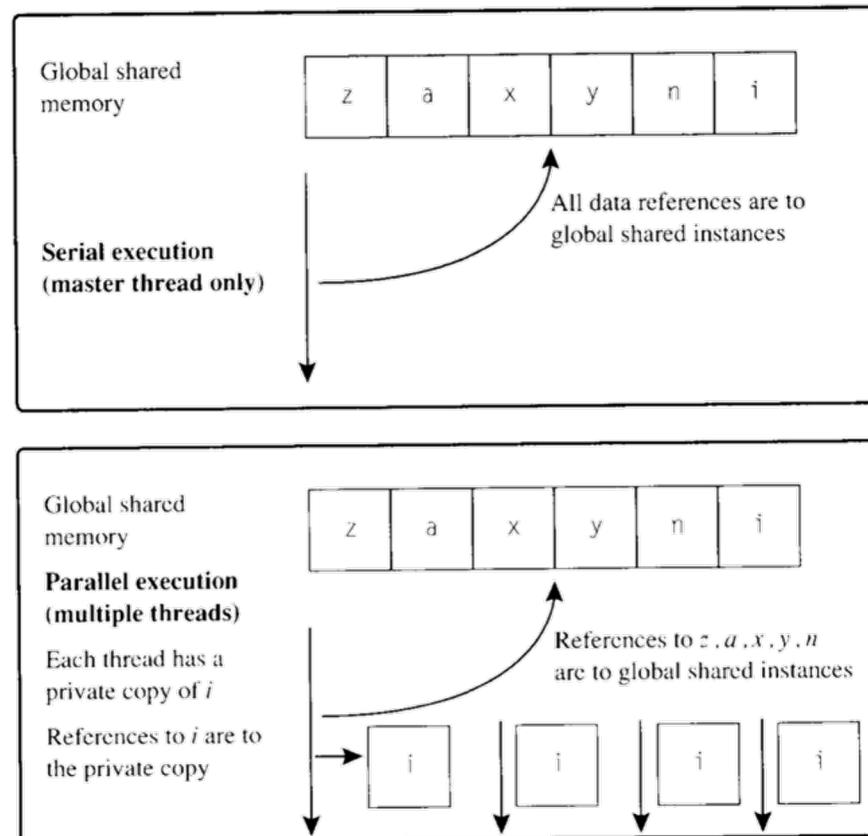


Figure 2.3

The behavior of private variables in an OpenMP program.

OpenMP - Scheduling

```
#pragma omp for schedule(static)
```



```
#pragma omp for schedule(static,3)
```



```
#pragma omp for schedule(dynamic)
```



```
#pragma omp for schedule(dynamic,2)
```



```
#pragma omp for schedule(guided)
```



```
#pragma omp for schedule(guided,2)
```



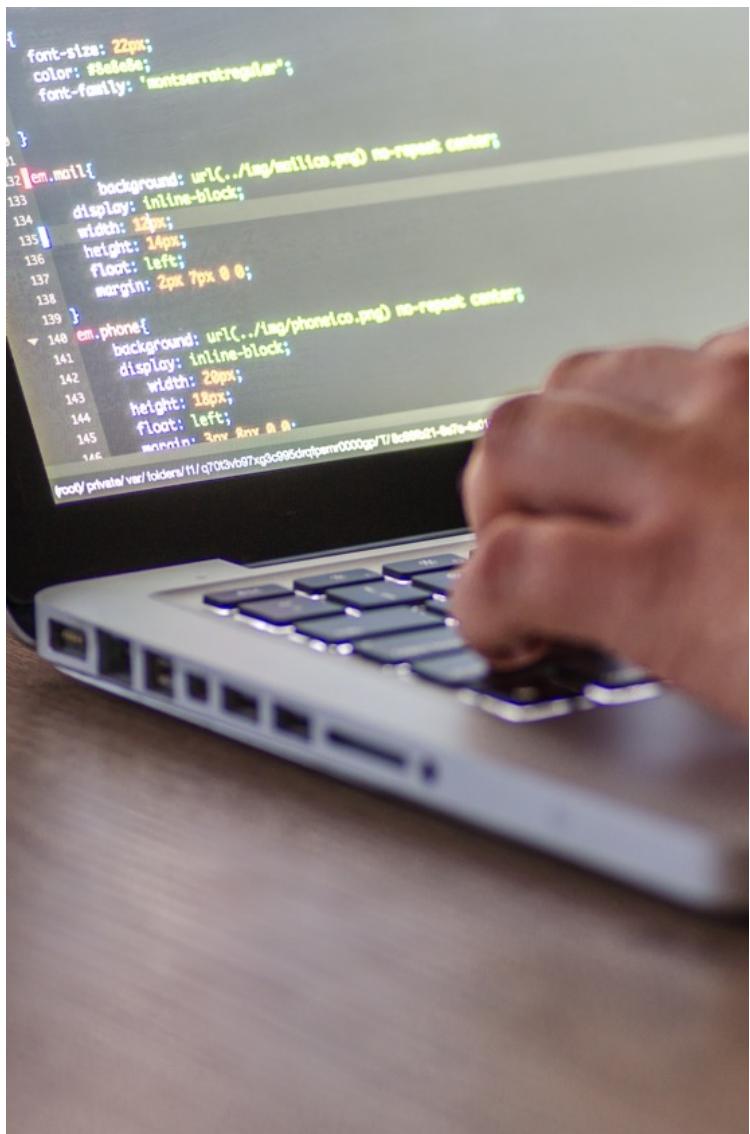
OpenMP – o conceito de redução

```
for(i=1; i<=n; i++){
    sum = sum + a[i];
}
```

```
#pragma omp parallel for reduction(+:sum)
{
    for(i=1; i<=n; i++){
        sum = sum + a[i];
    }
}
```

OpenMP - Demonstrações

- Demonstração em sala de aula de alguns códigos e suas otimizações com OpenMP





Insper Supercomputação



Aula - 12

- OpenMP

OpenMP

Sections
Tasks



Sections

- A diretiva sections divide o trabalho de forma não iterativa em seções separadas, aonde cada seção será executada por uma “thread” do grupo. Representa a implementação de paralelismo funcional, ou seja, por código.
- Algumas observações:
 - A diretiva sections define a seção do código sequencial onde será definida as seções independentes, através da diretiva section;
 - Cada section é executada por uma *thread* do grupo;
 - Existe um ponto de sincronização implícita no final da diretiva section, a menos que se especifique o atributo nowait;
 - Se existirem mais *threads* do que seções, o OpenMP decidirá, quais *threads* executarão os blocos de section, e quais, não executarão.

```
#include <omp.h>
#define N 1000
int main () {
int i, n=N;
float a[N], b[N], c[N];
for (i=0; i < N; i++) a[i] = b[i] = i * 1.0,
#pragma omp parallel shared(a,b,c,n) private(i) {
#pragma omp sections nowait {
#pragma omp section
for (i=0; i < n/2; i++)
    c[i] = a[i] + b[i];
#pragma omp section
for (i=n/2; i < n; i++)
    c[i] = a[i] + b[i];
} /* fim seções*/
} /* fim parallel */
}
```

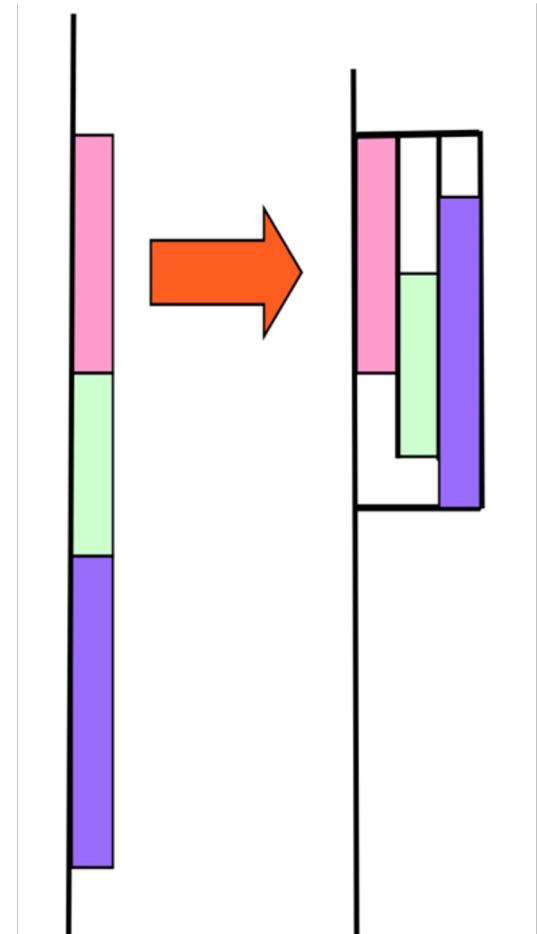
Definição de uma área de seções.

Primeira seção

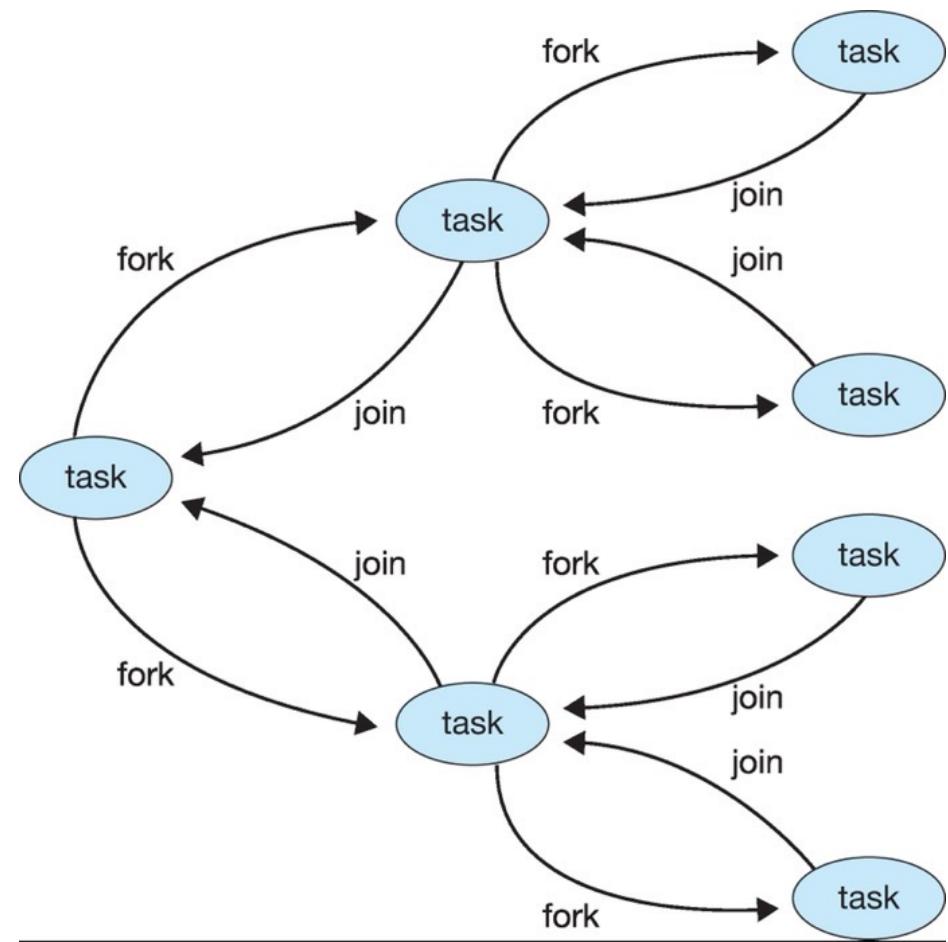
Segunda seção

O que são tarefas (tasks)?

- A tarefa é definida em um bloco estruturado de código
- Tarefas podem ser aninhadas: isto é, uma tarefa pode gerar outras novas tarefas
- Cada thread pode ser alocada para rodar uma tarefa
- Não existe ordenação no início das tarefas
- Tarefas são unidades de trabalho independentes

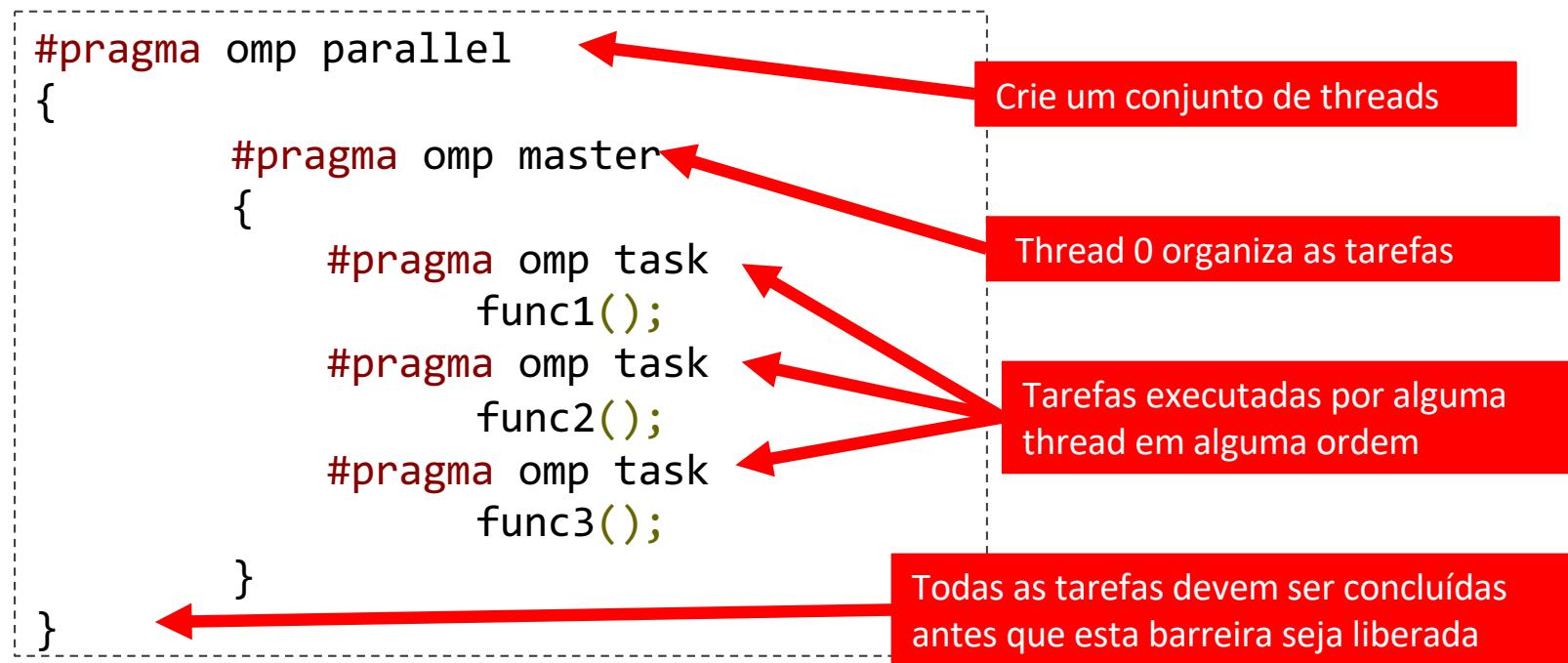


Tasks / Fork-Join



Tarefas em OpenMP

#pragma omp task[clauses]



Estrutura Padrão

```
#include <stdio.h>
#include <omp.h>
int main()
{ printf("I think");
#pragma omp parallel
{
#pragma omp single
{
#pragma omp task
printf(" car");
#pragma omp task
printf(" race");
}
}
printf("s");
printf(" are fun!\n");
}
```

Esperando (taskwait)

```
#pragma omp parallel
{
    #pragma omp single
    {
        #pragma omp task
        fred();
        #pragma omp task
        daisy();
        #pragma taskwait
        #pragma omp task
        billy();
    }
}
```

fred() and daisy()
must complete before
billy() starts

Fibonacci

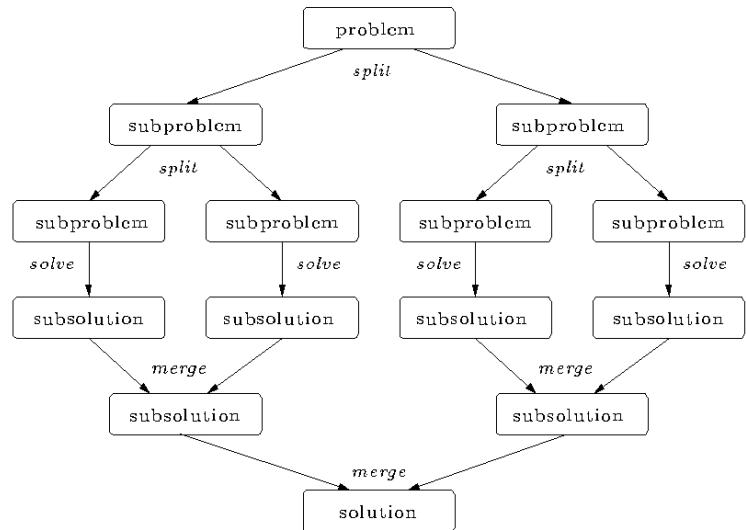
- É possível criar tarefas para esse problema?
- Altere a função fib para que ela chame uma task antes de calcular x e outra task antes de y.
- Lembre-se que quem chama fib (na função main) também precisa de uma task omp do tipo single (a task que dispara as outras tasks).
- Um ponto importante, se $n < 20$, calcule o Fibonacci sem o OpenMP.

```
#include<iostream>
#include<omp.h>
using namespace std;

int fib (int n) {
    int x, y;
    if(n<2) return n;
    x = fib(n-1);
    y = fib(n-2);
    return (x+y);
}

int main() {
    int Nw = 1000;
    float time = omp_get_wtime();
    fib(Nw);
    time = omp_get_wtime() - time;
    cout << "Tempo em segundos : " << time << endl;
}
```

Resolução



```
#include<iostream>
#include<omp.h>
#include <math.h>
using namespace std;

int fib (int n) {
    int x, y;
    if(n<2) return n;
    if (n < 20) {
        return fib(n-1) + fib(n-2);
    } else {
#pragma omp task shared(x)
x = fib(n-1);
#pragma omp task shared(y)
y = fib(n-2);
#pragma omp taskwait
return x+y;
    }
}

int main() {
    int NW = 50;
    float time;
    time = omp_get_wtime();
#pragma omp parallel
{
#pragma omp single
fib(NW);
}
time = omp_get_wtime() - time;
cout << "Tempo em segundos : " << time << endl;
}
```

OpenMP – Roteiro

- Seguir o roteiro da aula



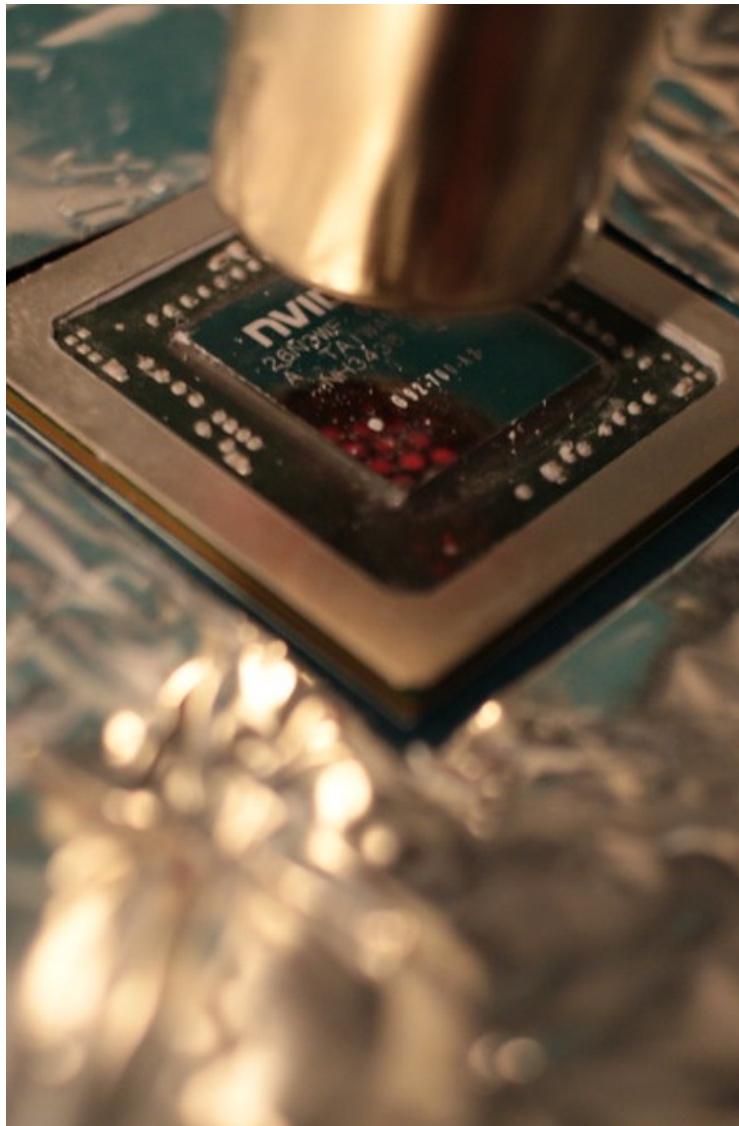
Mochila Binária - OpenMP



- Sua tarefa:
- Adaptar o código da mochila recursiva por busca exaustiva, para suportar OpenMP.
- Avalie o desempenho



Insper Supercomputação



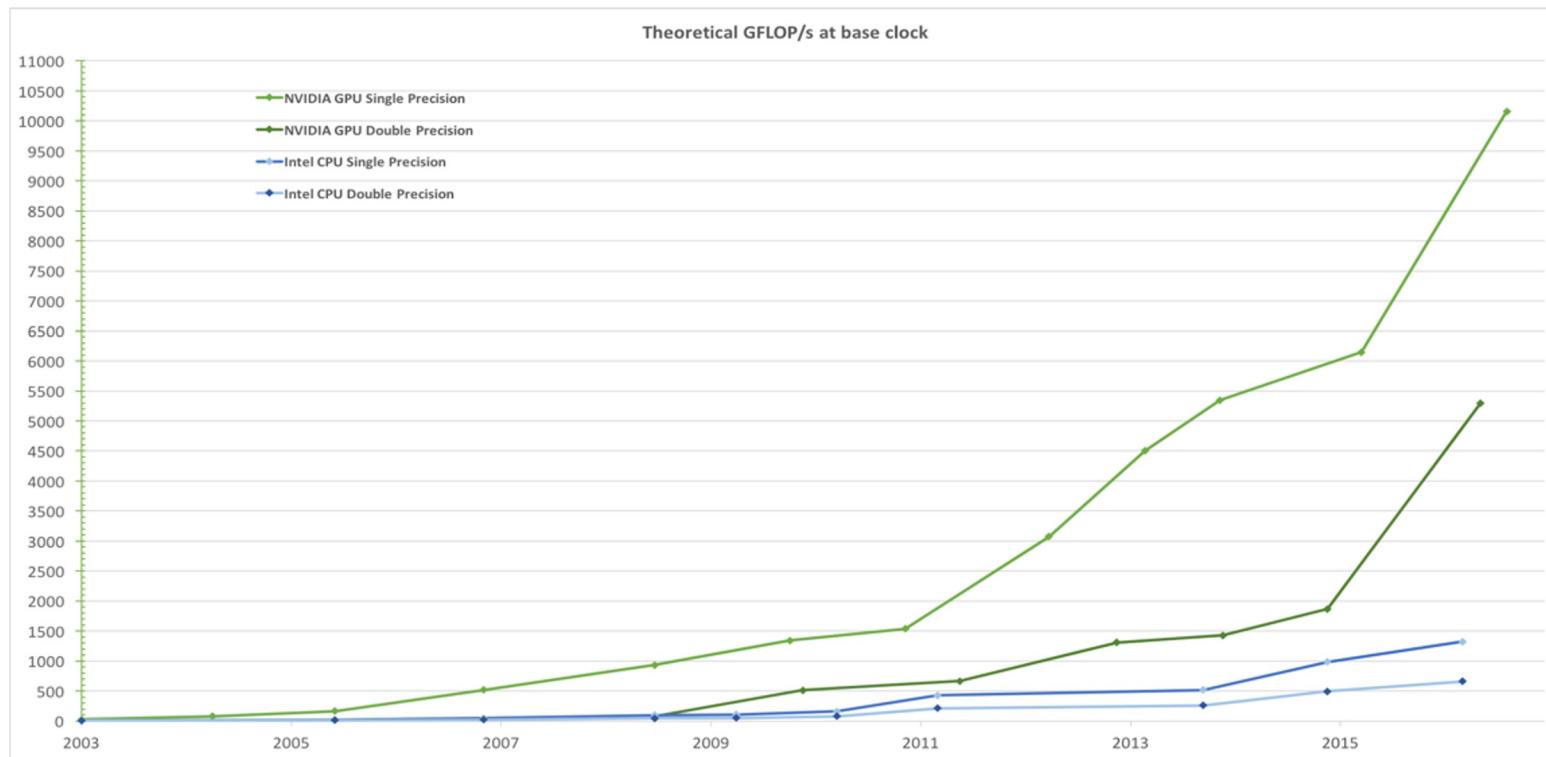
Aula - 13

- Introdução a GPU

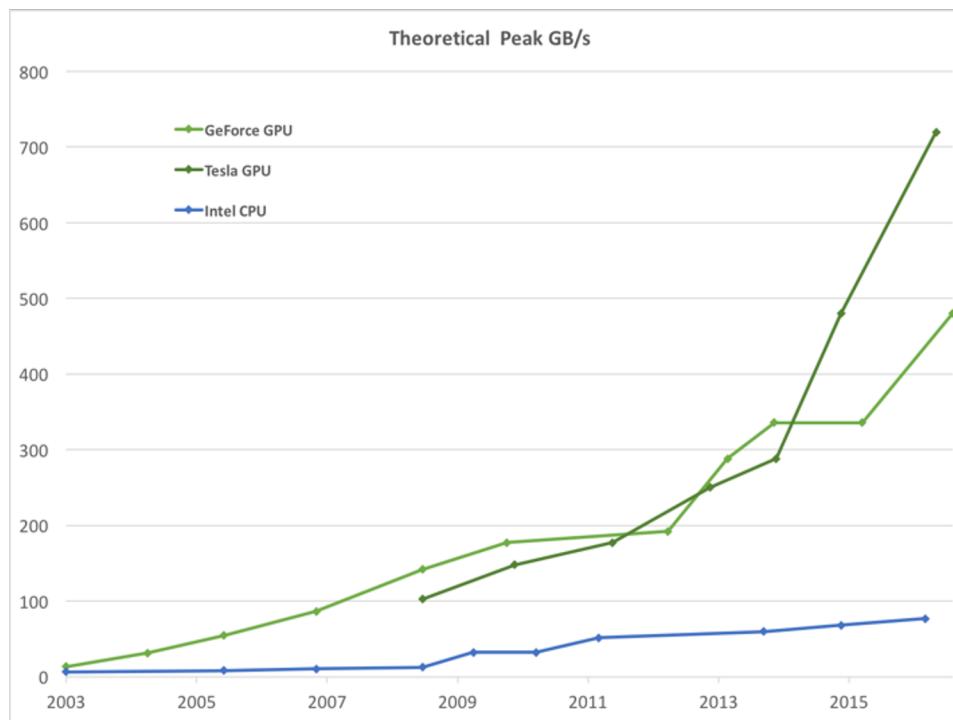
Nossos objetivos

- Diferenciar dispositivos de latência (CPUs) e de throughput (GPUs)
- Compreender o layout de memória e transferência de dados em sistemas heterogêneos (CPU \Leftrightarrow GPU)
- Compilar primeiros programas na GPU

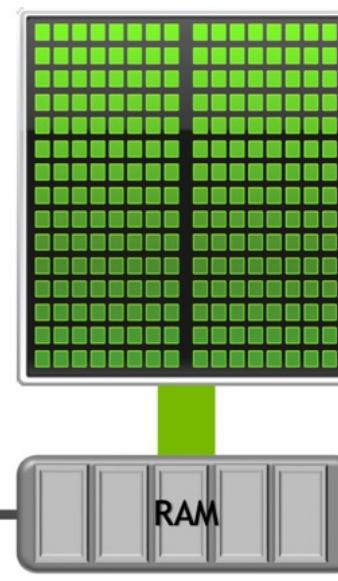
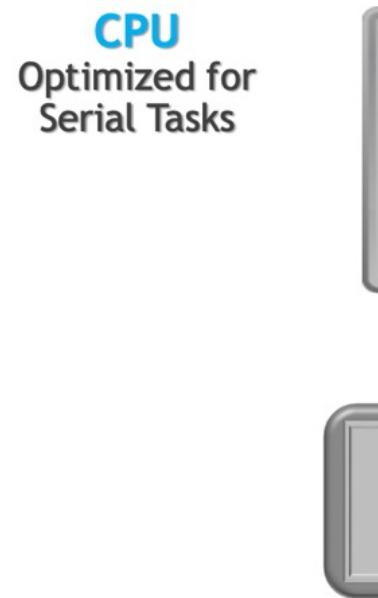
Desempenho em GFLOPS



Desempenho em GB/s



CPUs e GPUs



Speed vs Throughput

Speed



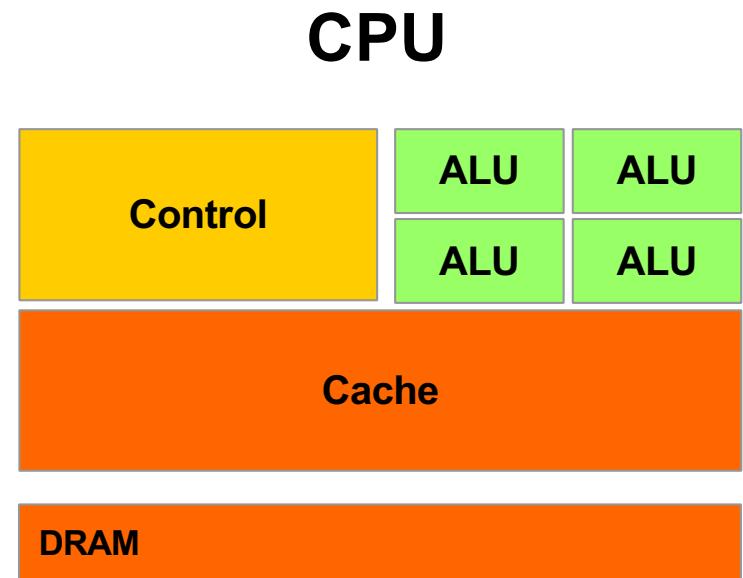
Throughput



Which is better depends on your needs...

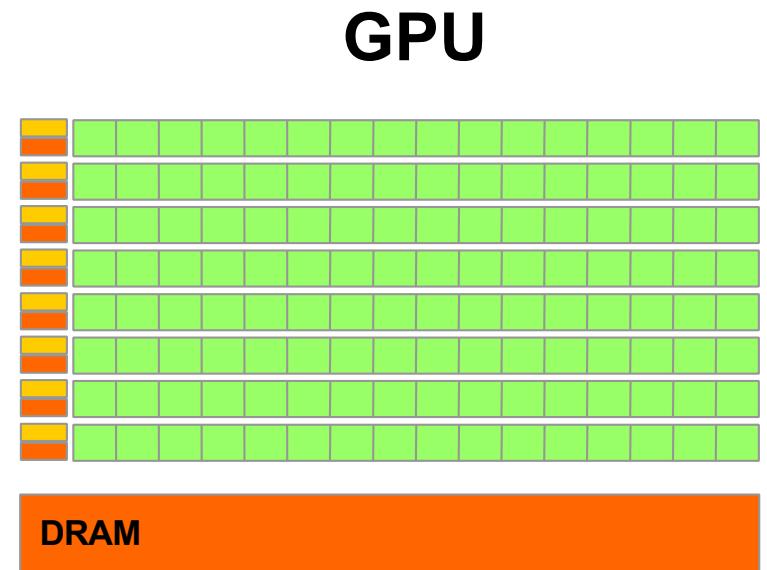
A CPU minimiza latência

- ULA potente, minimiza latência das operações
- Cache grande:
 - Acelera operações lentas de acesso à RAM
 - Mas cache-misses são custosos
 - Baixa performance / watt



A GPU maximiza throughput

- ULA simples
 - Eficiente energeticamente
 - Alta taxa de transferência
- Cache pequeno
 - Acesso contínuo a RAM
- Controle simples
- Número massivo de threads



CPU vs GPU

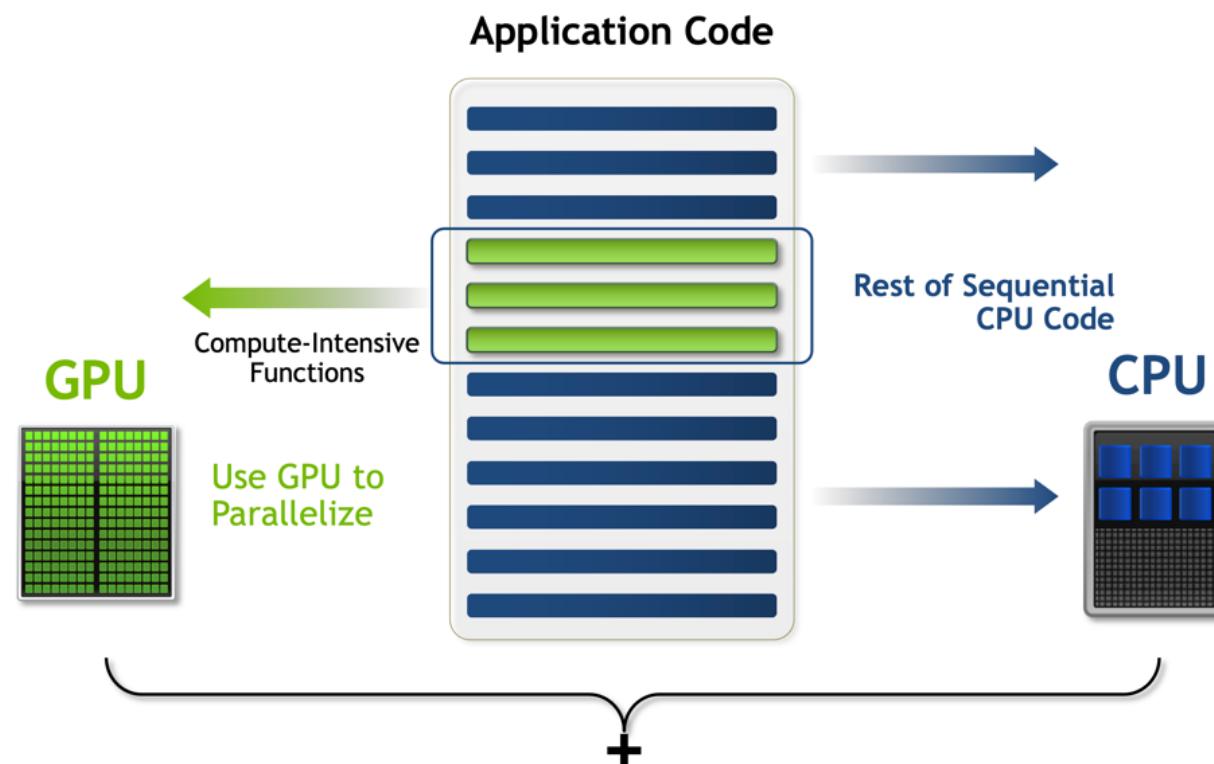
- CPUs para partes sequenciais onde uma latência mínima é importante
 - CPUs podem ser 10X mais rápidas que GPUs para código sequencial



- GPUs para partes paralelas onde a taxa de transferência(throughput) bate a latência menor.
 - GPUs podem ser 10X mais rápidas que as CPUs para código paralelo

CPU vs GPU

Minimum Change, Big Speed-up



Como usar a GPU?

Aplicações

Bibliotecas

Diretivas de
Compilação

Linguagens de
Programação

Fácil de Usar
Alto Desempenho

Simples de Usar
Portabilidade de Código

Maior Desempenho
Maior Flexibilidade

Bibliotecas aceleradas por GPU

Álgebra Linear
FFT, BLAS,
SPARSE, Matrix



cULA tools

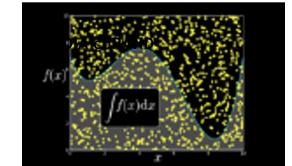


CUSP

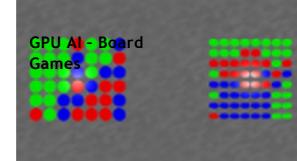
Numéricas/Math
RAND, Estatísticas



ArrayFire



Estrutura de Dados/IA
Sort, Scan, Zero Sum



Processamento Visual
Imagem & Video



Sundog™
Software

Bibliotecas aceleradas por GPU

- Facilidade de uso: O uso de bibliotecas permite a aceleração da GPU sem conhecimento aprofundado da programação da GPU
- Simplicidade: Muitas bibliotecas aceleradas por GPU seguem APIs padrão, permitindo aceleração com mudanças mínimas de código
- Qualidade: As bibliotecas oferecem implementações de alta qualidade de funções encontradas em uma ampla variedade de aplicativos

Linguagens de Programação

Numerical analytics



MATLAB, Mathematica, LabVIEW

Fortran



CUDA Fortran

C



CUDA C

C++



CUDA C++

Python



PyCUDA, Copperhead, Numba

Programando para GPU

- Compilador especial: nvcc
- Endereçamento de memória separado
 - Dados precisam ser copiados de/para GPU
 - Isto leva tempo
- Funções especiais (kernels) para rodar na GPU

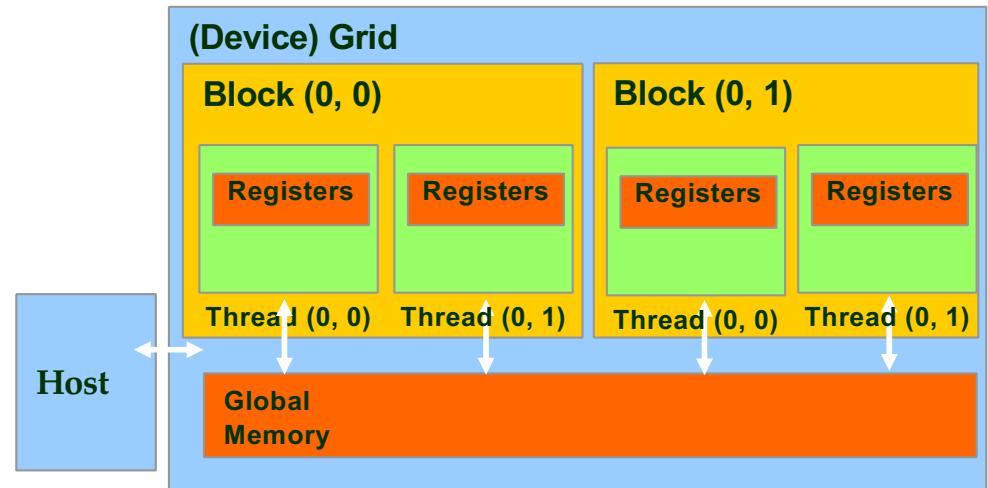
Memória em GPUs

Código da GPU (device) pode:

- Cada thread ler e escrever nos **registradores**
- Ler e escrever na **memória global**

Código da CPU (host) pode:

- Transferir dados de e para **memória global**

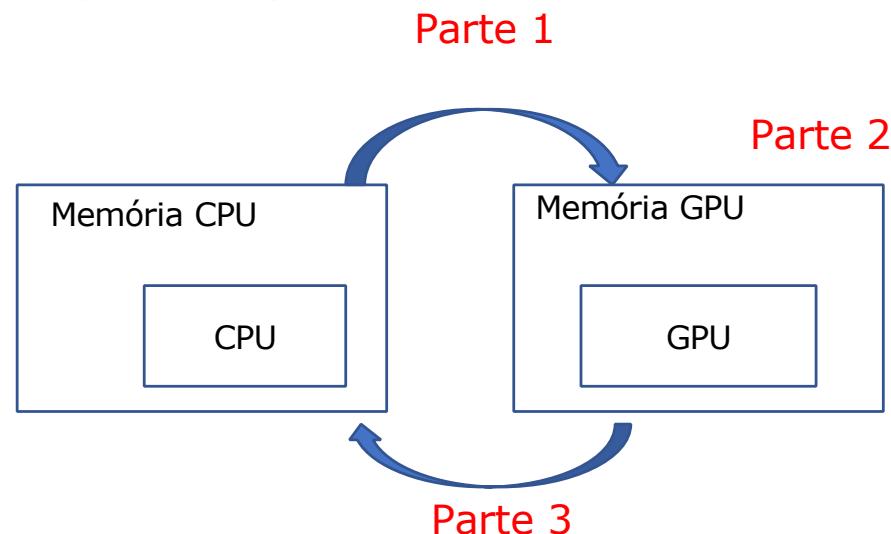


Fluxo dos programas

Parte 1: copia dados CPU → GPU

Parte 2: processa dados na GPU

Parte 3: copia resultados GPU → CPU



Biblioteca que usaremos - Thrust



Importante: infra

Se você tem uma GPU:

- pode usá-la diretamente na disciplina;
- instale o pacote nvidia-cuda-toolkit e os drivers compatíveis

Se você não tem GPU:

- compile código usando thrust/OpenMP
- solicite a conversão da sua VM para GPU com o Tiago (tiagoaodc@insper.edu.br)

A partir da próxima semana vamos supor que todos já tem acesso a uma GPU com compilador funcionando.

Nvidia Thrust

Vantagens:

- Simplifica transferências de memória
- Duas operações customizáveis (*reduce, transform*)
- Suporta OpenMP e CUDA

Desvantagens:

- Limitado: menos recursos e desempenho que CUDA C
- Só tem dois tipos de operações
- Baseado em *templates* – *difícil de debugar erros de compilação*

Nvidia Thrust – tipos de dados

Apenas dois tipos

`thrust::device_vector<T>`

- Vetor genérico de dados na GPU
- Automaticamente alocado e desalocado
- Cópia é feita usando atribuição

`thrust::host_vector<T>`

- Vetor genérico de dados na CPU
- Pode ser substituído em vários lugares por containers da STL ou ponteiros “normais”

Thrust - Exemplo

```
thrust::host_vector<double> vec_cpu(10); // alocado na CPU  
  
vec1[0] = 20;  
vec2[1] = 30;  
  
thrust::host_vector<double> vec_gpu (10); // alocado na GPU  
  
vec_gpu = vec_cpu; // copia o conteúdo da CPU para GPU  
thrust::device_vector<double> vec2_gpu (vec_cpu); // também transfere para GPU
```

Thrust - Iteradores

Funcionam igual aos iteradores de std::vector

```
v.begin() // primeiro elemento
```

```
v.end() // último elemento
```

```
v.begin()+2 // v[2]
```

```
i = v.begin() + 3; *i = 4; // v[3] = 4
```

Thrust - Exemplo

```
thrust::device_vector<int> v(5, 0); // vetor de 5 posições zerado
// v = {0, 0, 0, 0, 0}
thrust::sequence(v.begin(), v.end()); // inicializa com 0, 1, 2, ....
// v = {0, 1, 2, 3, 4}
thrust::fill(v.begin(), v.begin() + 2, 13); // dois primeiros elementos = 3
// v = {13, 13, 2, 3, 4}
```

Thrust – Redução

Resume o vetor para um escalar

- Soma todos elementos
- Máximo/mínimo do vetor
- Contagens
- Suporta iteradores, o que torna a operação bastante flexível.

Thrust - Exemplo

```
val = thrust::reduce(iter_comeco, iter_fim, inicial, op);
// iter_comeco: iterador para o começo dos dados
// iter_fim: iterador para o fim dos dados
// inicial: valor inicial
// op: operação a ser feita.
```

Thrust - Transformação

Operações elemento a elemento entre pares de vetores ou um só vetor.

- Aritmética ponto a ponto
- Permite criação de operações customizadas
- Suporta iteradores de entrada e saída
- Funciona também para operações locais (imagens)

Thrust - Exemplo

```
thrust::device_vector<double> V1(10, 0);
thrust::device_vector<double> V2(10, 0);
thrust::device_vector<double> V3(10, 0);
thrust::device_vector<double> V4(10, 0);
// inicializa V1 e V2 aqui

// soma V1 e V2
thrust::transform(V1.begin(), V1.end(), V2.begin(), V3.begin(), thrust::plus<double>());

// multiplica V1 por 0.5
thrust::transform(V1.begin(), V1.end(),
                  thrust::constant_iterator<double>(0.5),
                  V4.begin(), thrust::multiplies<double>());
```

Thrust - Roteiro

Iniciando com Thrust

- Alocação de memória em CPU e GPU
- Utilização das operações de redução e transformação
- Operações elemento a elemento disponíveis na thrust



Five Operations You Can Do with a Lot of Data in Parallel



- Generate/Create
 - Automatically fill with programmatically defined data
- Transform
 - Apply some “operation” to each element of the data
 - Also called “Map” in many contexts
- Compact
 - Take only the elements in which you are interested
 - Also called “Filter” in many contexts
- Expand
 - The opposite of Compact
 - Create a larger data set from a smaller data set
- Aggregate
 - Calculate a “summary” of your data (e.g., sum or average)
 - Also called “Reduce” or “Fold”
 - “Scan” also provides all intermediate values

Simple Examples with Thrust Pseudocode

- **Generate**

```
thrust::sequence(0, 4) 0 1 2 3 4
```

- **Transform**

input	4	5	2	1	3
thrust::transform(+1)	5	6	3	2	4

- **Compact**

input	4	5	2	1	3
thrust::copy_if(even)	4	2			

- **Expand**

input	4	5	2	1	3					
thrust::for_each(x, 2x)	4	8	5	10	2	4	1	2	3	6

- **Aggregate**

input	4	5	2	1	3
-------	---	---	---	---	---

thrust::reduce(+)	15
-------------------	----

Generate Data in Parallel

- Many copies of a certain constant value

- // Ten elements with initial value of integer 1
thrust::device_vector<int> x(10, 1);

- A sequence of increasing or decreasing values

- // Allocate space for ten integers, uninitialized
thrust::device_vector<int> y(10);
// Fill the space with integers
thrust::sequence(y.begin(), y.end(), 5, 2);

- Random Values

- Multiple copies of a random number generator
 - Give each one a different seed

Transform: Vector Addition

Apply a binary operator “plus” to each element in x and y

- ```
thrust::transform(x.begin(), x.end(), // begin and end of the
 // first input vector
 y.begin(), // begin of the second
 // input vector
 result.begin(), // begin of the result
 // vector
 thrust::plus<int>()); // predefined integer
 // addition
```
- |         |   |   |    |    |    |    |    |    |    |    |
|---------|---|---|----|----|----|----|----|----|----|----|
| x:      | 1 | 1 | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  |
|         | + |   |    |    |    |    |    |    |    |    |
| y:      | 5 | 7 | 9  | 11 | 13 | 15 | 17 | 19 | 21 | 23 |
|         | = |   |    |    |    |    |    |    |    |    |
| result: | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 | 22 | 24 |

# Transform: Uniform Sampling of a Mathematical Function

- Q: How are we going to generate something more complicated?  
A: Start from some sequence and apply some transformation
- Sampling the function  $f(x) = x^2$  in the interval of  $[0, 1]$

```
- // Generate a sequence of x_i in [0,1] with $dx=0.1$
// in between each of them
float dx = 1.0f/10.0f;
thrust::sequence(x.begin(), x.end(), 0.0f, dx);

// Apply the square operation to each of the x_i
// to transform into $f(x_i) = y_i = x_i^2$
thrust::transform(x.begin(), x.end(),
 y.begin(),
 square());
```

Inspe

|    |     |      |      |      |      |      |      |      |      |      |     |
|----|-----|------|------|------|------|------|------|------|------|------|-----|
| x: | 0.0 | 0.1  | 0.2  | 0.3  | 0.4  | 0.5  | 0.6  | 0.7  | 0.8  | 0.9  | 1.0 |
| y: | 0.0 | 0.01 | 0.04 | 0.09 | 0.16 | 0.25 | 0.36 | 0.49 | 0.64 | 0.81 | 1.0 |

# Simple Numerical Integration: Example

```
thrust::device_vector<int> width(11, 0.1);
width = 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1

thrust::sequence(x.begin(), x.end(), 0.0f, 0.1f);
x = 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

thrust::transform(x.begin(), x.end(), height.begin(), square());
height = 0.0 0.01 0.04 0.09 0.16 0.25 0.36 0.49 0.64 0.81 1.0

thrust::transform(width.begin(), width.end(), height.begin(), area.begin(),
thrust::multiplies<float>())
area = 0.0 0.001 0.004 0.009 0.016 0.025 0.036 0.049 0.064 0.081 0.1

total_area = thrust::reduce(area.begin(), area.end());
total_area = 0.385

thrust::inclusive_scan(area.begin(), area.end(), accum_areas.begin());
accum_areas = 0.0 0.001 0.005 0.014 0.030 0.055 0.091 0.140 0.204 0.285 0.385
```

Inspect



# Reduce: Simple Numerical Integration

Apply what we learned to estimate the area under a curve

Create a constant vector of widths

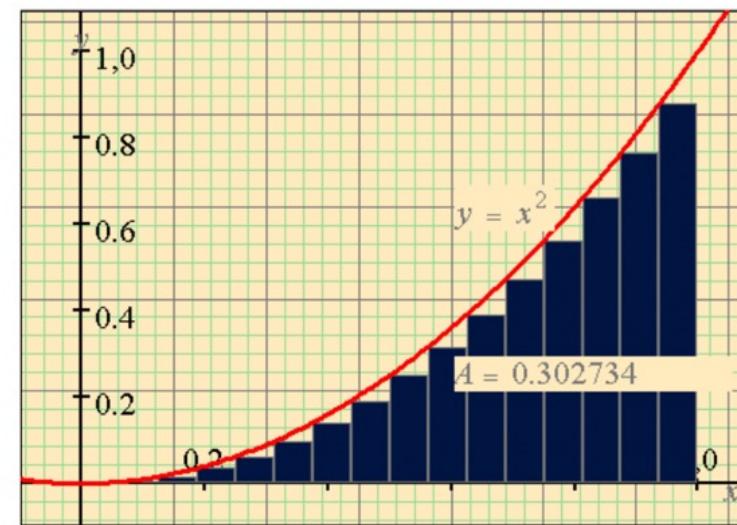
Create a vector of heights from the function values

Apply multiply operation on each element of width and height

Sum all the computed areas to get the total area

In calculus, this is a method of estimating the integral

Inspire  
Los Alamos



$$\int_0^1 f(x)dx \approx \sum_{i=1}^n f(x_i)\Delta x$$

# Scan: Simple Numerical Integration

- What happens if we are interested in the integral  $F(t) = F(0) + \int_0^t f(x)dx$  on the interval  $[0, 1]$  instead of just a number?
- Calculate a running sum by using scan
- ```
thrust::inclusive_scan(y_dx.begin(), y_dx.end(),
                      F.begin());
```
- $f(x_i) * dx = 0.0 0.001 0.004 0.009 0.016 0.025 0.036 0.049 0.064 0.081 0.1$
 $F(t) = 0.0 0.001 0.005 0.014 0.030 0.055 0.091 0.140 0.204 0.285 0.385$
- The last element of the scan (0.385) is the same as the output of reduce
- In mathematical terms,

$$\int_0^1 f(x)dx = F(1) - F(0)$$

Insp



```

1 #include <thrust/device_vector.h>
2 #include <thrust/iterator/constant_iterator.h>
3 #include <thrust/scan.h>
4 #include <thrust/sort.h>
5
6 struct saxpy_functor
7 {
8     const float a;
9     saxpy_functor(float _a) : a(_a) {}
10
11     __host__ __device__
12     float operator()(const float& x, const float& y) const
13     {
14         return a * x + y;
15     }
16 };
17
18 int main(void)
19 {
20     // Declare host and device vectors
21     thrust::host_vector<int> H1(5), H2(5), H3;
22     thrust::device_vector<int> D1, D2, D3;
23
24     // Initialize H1 with [2,2,2,2,2] and H2 with [0,1,2,3,4]
25     thrust::fill(H1.begin(), H1.end(), 2);
26     thrust::sequence(H2.begin(), H2.end(), 0);
27
28     // Copy data from host vectors to device vectors
29     D1 = H1; D2 = H2;
30
31     // Add D1 and D2 and store in D1
32     thrust::transform(D1.begin(), D1.end(), D2.begin(), D1.begin(), thrust::plus<int>());
33
34     // H1 = D1 = [2,3,4,5,6]
35     H0 = D1; std::cout << "H0: "; for (int i=0; i<H0.size(); i++) std::cout << H0[i] << " "; std::cout << std::endl;
36
37     // Perform inclusive scan (prefix sum) on D1 and store in D2
38     thrust::inclusive_scan(D1.begin(), D1.end(), D2.begin());
39
40     // H2 = D2 = [2,5,9,14,20]
41     H2 = D2; std::cout << "H2: "; for (int i=0; i<H2.size(); i++) std::cout << H2[i] << " "; std::cout << std::endl;
42
43     // Use a permutation iterator to gather elements from D1 based on specified indices in D2, storing in H1
44     D2[0] = 1; D2[1] = 4; D2[2] = 2; D2[3] = 0; D2[4] = 3;
45     thrust::copy(thrust::make_permutation_iterator(D1.begin(), D2.begin()), thrust::make_permutation_iterator(D1.begin(), D2.end()), H1.begin());
46
47     // H1 = [3,6,4,2,5]
48     std::cout << "H1: "; for (int i=0; i<H1.size(); i++) std::cout << H1[i] << " "; std::cout << std::endl;
49
50     // Compute saxpy (A*x + y) where x is D1, y is [3,3,3,3] and A is 2, storing result in D2
51     thrust::transform(D1.begin(), D1.end(), thrust::make_constant_iterator(3), D2.begin(), saxpy_functor(2));
52
53     // H2 = D2 = [7,9,11,13,15]
54     H2 = D2; std::cout << "H2: "; for (int i=0; i<H2.size(); i++) std::cout << H2[i] << " "; std::cout << std::endl;
55
56     // Copy H1 to D1, fill D3 with [0,1,2,3,4], and sort D1, D2, and D3 based on values in D1
57     D1 = H1;
58     D3.resize(5); copy(thrust::make_counting_iterator(0), thrust::make_counting_iterator(0)+5, D3.begin());
59     thrust::sort_by_key(D1.begin(), D1.end(), thrust::make_zip_iterator(thrust::make_tuple(D2.begin(), D3.begin())));
60
61     // H1 = D1 = [2,3,4,5,6]; H2 = D2 = [13,7,11,15,9]; H3 = D3 = [3,0,2,4,1]
62     H1 = D1; std::cout << "H1: "; for (int i=0; i<H1.size(); i++) std::cout << H1[i] << " "; std::cout << std::endl;
63     H2 = D2; std::cout << "H2: "; for (int i=0; i<H2.size(); i++) std::cout << H2[i] << " "; std::cout << std::endl;
64     H3 = D3; std::cout << "H3: "; for (int i=0; i<H3.size(); i++) std::cout << H3[i] << " "; std::cout << std::endl;
65
66     // Compute sum of elements in D2; sum = 55
67     int sum = thrust::reduce(D2.begin(), D2.end());
68     std::cout << "Sum of D2: " << sum << std::endl;
69
70     return 0;
71 }

```

> nvcc examples.cu -o examples
 > ./examples

- FAZER
 ESSE
 CODIGO
 JUNTO COM
 OS ALUNOS

```

#include <thrust/transform_reduce.h>
#include <thrust/functional.h>
#include <thrust/device_vector.h>
#include <thrust/host_vector.h>
#include <cmath>

// square<T> computes the square of a number f(x) -> x*x
template <typename T>
struct square
{
    __host__ __device__
    T operator()(const T& x) const {
        return x * x;
    }
};

int main(void)
{
    // initialize host array
    float x[4] = {1.0, 2.0, 3.0, 4.0};

    // transfer to device
    thrust::device_vector<float> d_x(x, x + 4);

    // setup arguments
    square<float> unary_op;
    thrust::plus<float> binary_op;
    float init = 0;

    // compute norm
    float norm = std::sqrt(thrust::transform_reduce(d_x.begin(),
                                                    d_x.end(),
                                                    unary_op,
                                                    init,
                                                    binary_op));

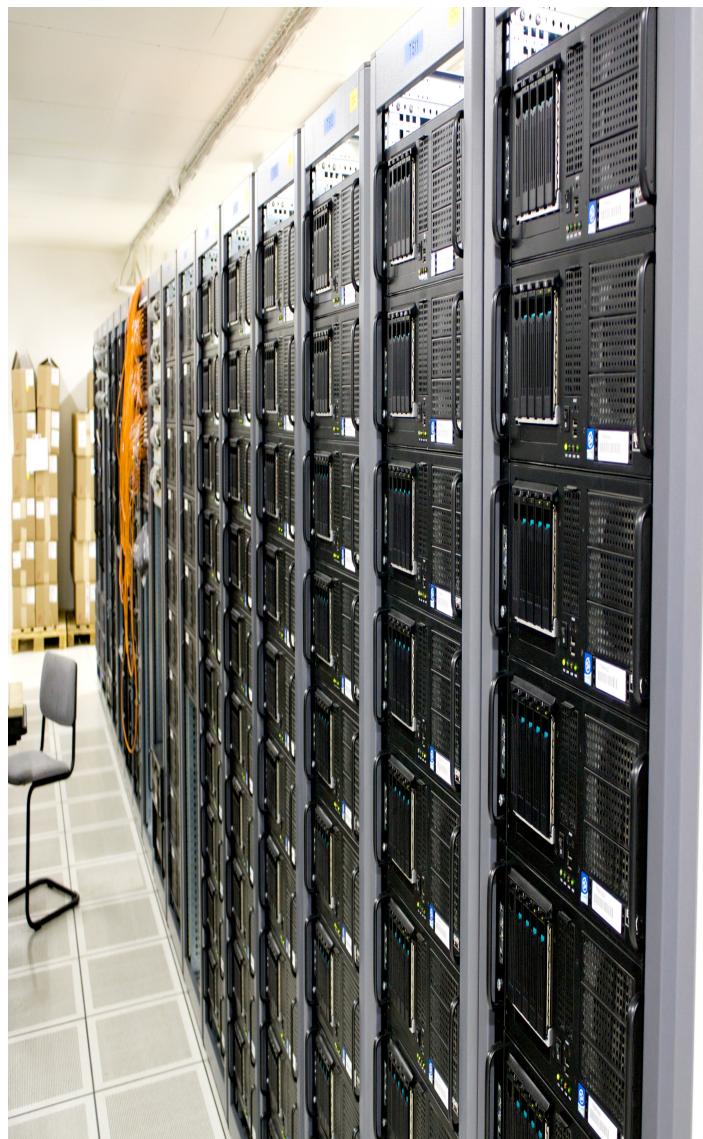
    std::cout << norm << std::endl;

    return 0;
}

```



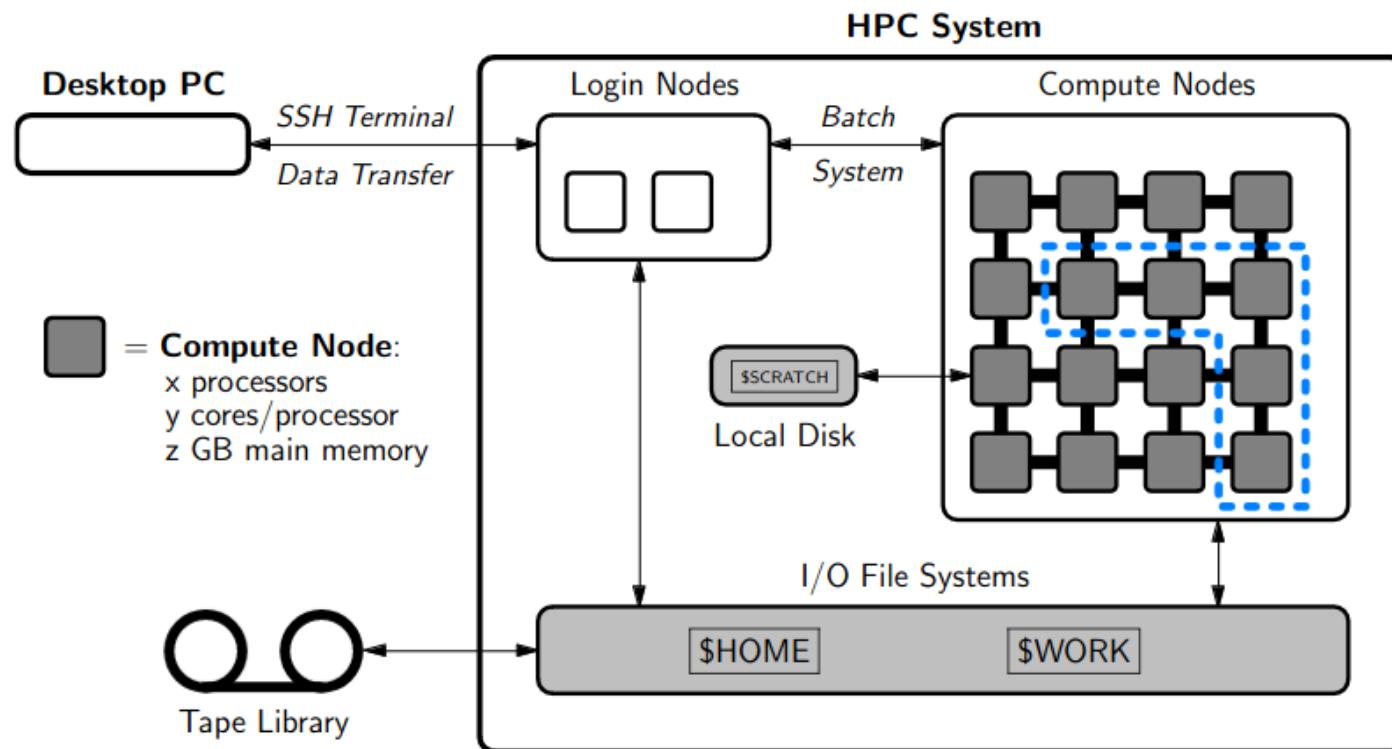
Insper Supercomputação



Aula

- Administração de Clusters com SLURM

Um típico ambiente de HPC



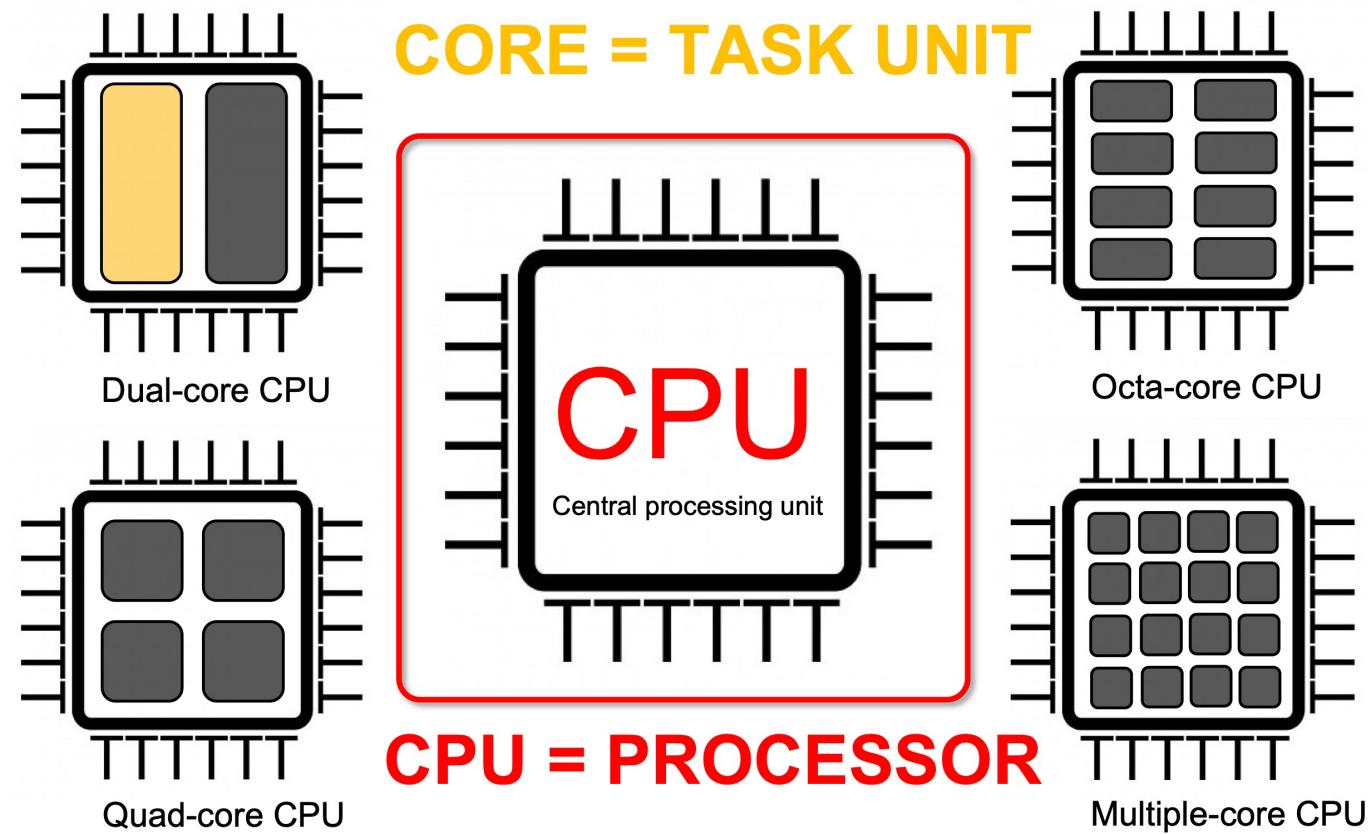
Fonte: <https://www.rz.uni-kiel.de/de/angebote/hiperf/hpc-course-12feb2020>

Antes de falarmos de SLURM, vamos recapular

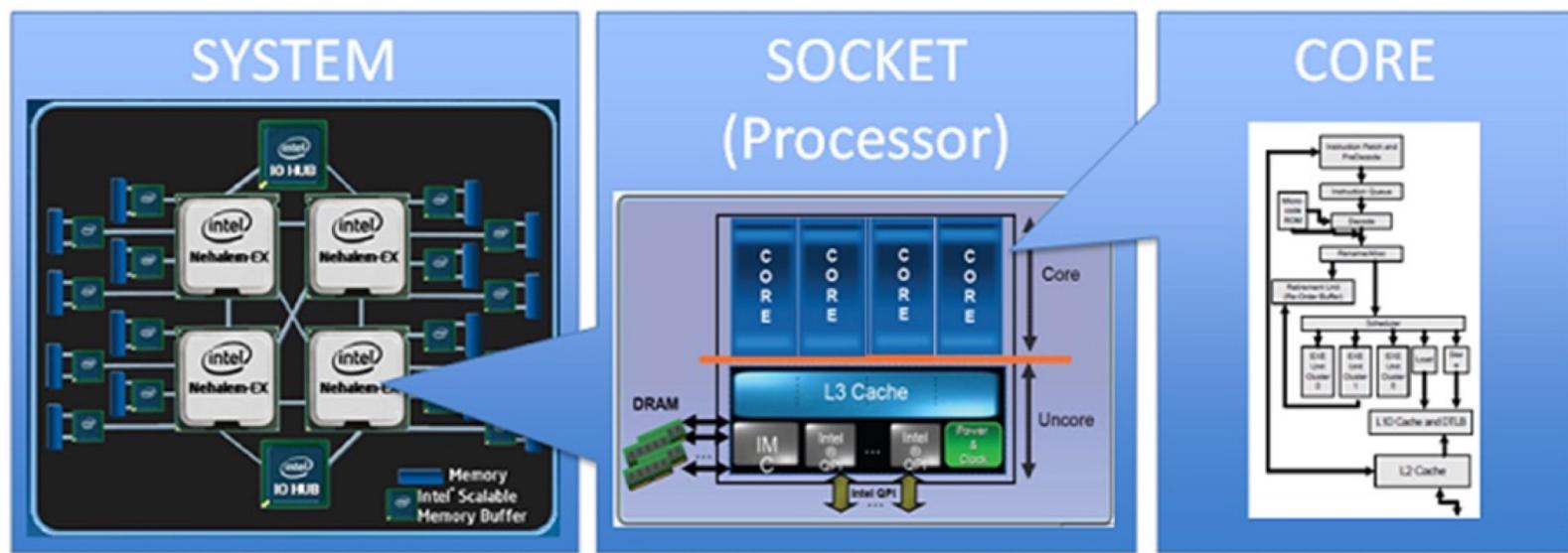
- Cores x CPU
- Como se avalia o desempenho de computadores (FLOPS)
- Clusters – como são



Cores x CPU



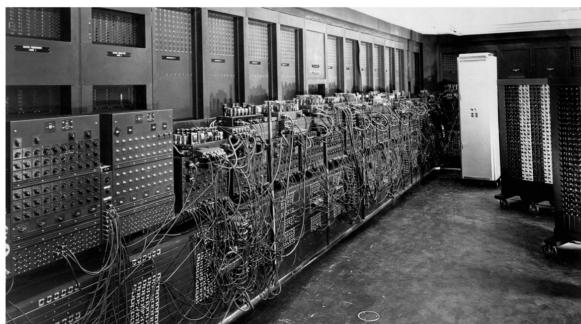
Cores x CPU



4 cores * 4 processors = 16 total cores

Medição de desempenho

- O desempenho dos ambientes de HPC é mensurado em **Floating Point Operations Per Second (FLOPS)**



ENIAC FLOPS: 500

- Por exemplo, o primeiro desktop PC a ter desempenho em teraflops (10^{12} FLOPS), foi em 2017, o Intel i97980XE

CPU clock rate: 4.4 GHz
CORE: 18 cores
FLOPs per cycle: 16



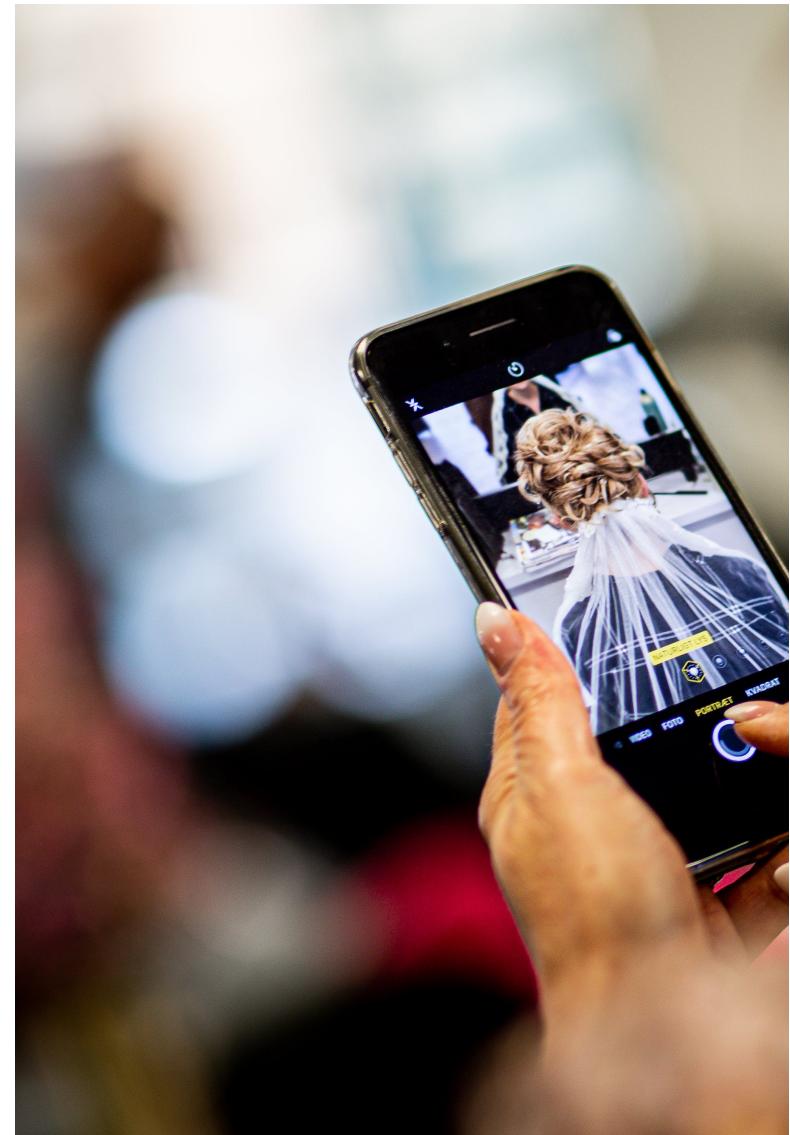
$$FLOPS = \text{cores} \times \text{clock} \times \frac{FLOPs}{cycle}$$

↓ ↓ ↓ ↓
1267 GHz 18 4.4 GHz 16
1.27 TFLOPS

Computer performance	
Name	FLOPS
yottaFLOPS	1024
zettaFLOPS	1021
exaFLOPS	1018
petaFLOPS	1015
teraFLOPS	1012
gigaFLOPS	109
megaFLOPS	106
kiloFLOPS	103

Medição de desempenho

- Já te falaram que você anda com um supercomputador no bolso?
- Vamos comparar um smartphone atual com um supercomputador de mais de 20 anos
 - Chip Apple A16 Bionic, usado no iPhone 14 Pro – 17 TFLOPS
 - Ano de 1996: Fujitsu 105MHz: 0.3 TFLOPS, com 167 cores
 - Anos 2000: ASCI WHITE, SP POWER3 375MHz: 7.3 TFLOPS, com 8.192 cores



TOP 500 – Junho 2023



Na foto, Frontier, em ORNL (US), o maior supercomputador da atualidade

1.1 exaflops ~ 10^{18} cálculos por segundo

Insper

www.insper.edu.br

R_{max} and R_{peak} values are in PFlop/s. For more details about other fields, check the TOP500 description.

R_{peak} values are calculated using the advertised clock rate of the CPU. For the efficiency of the systems you should take into account the Turbo CPU clock rate where it applies.

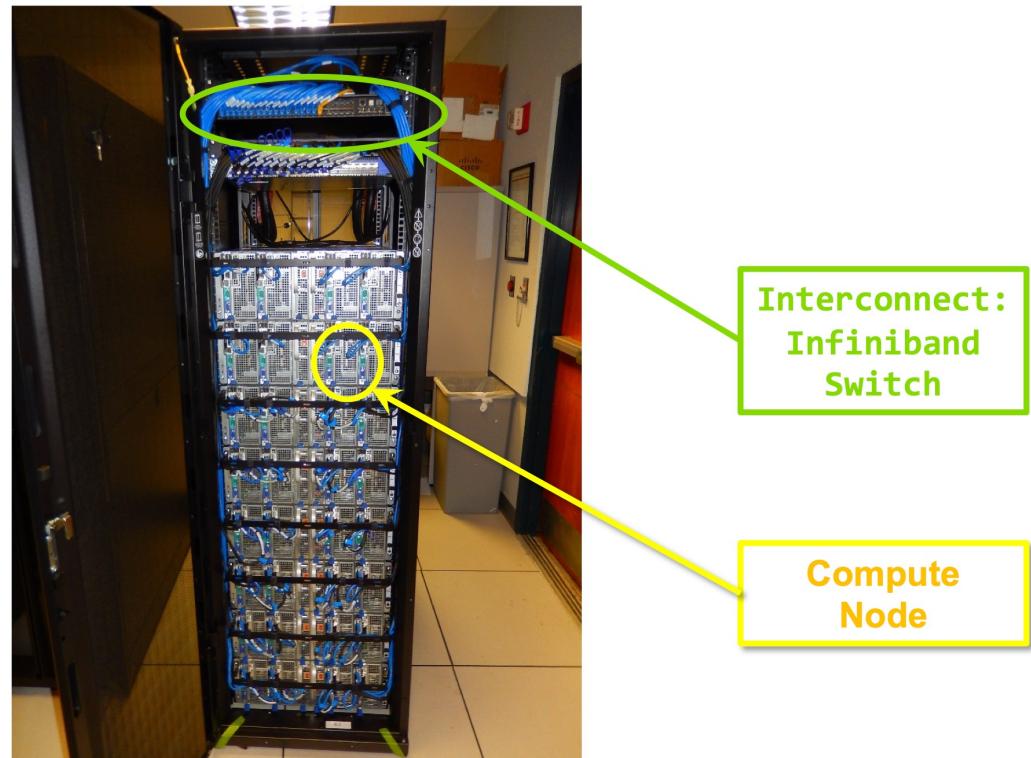
Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	Frontier - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,194.00	1,679.82	22,703
2	Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
3	LUMI - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,220,288	309.10	428.70	6,016
4	Leonardo - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, Atos EuroHPC/CINECA Italy	1,824,768	238.70	304.47	7,404
5	Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148.60	200.79	10,096

Um típico cluster



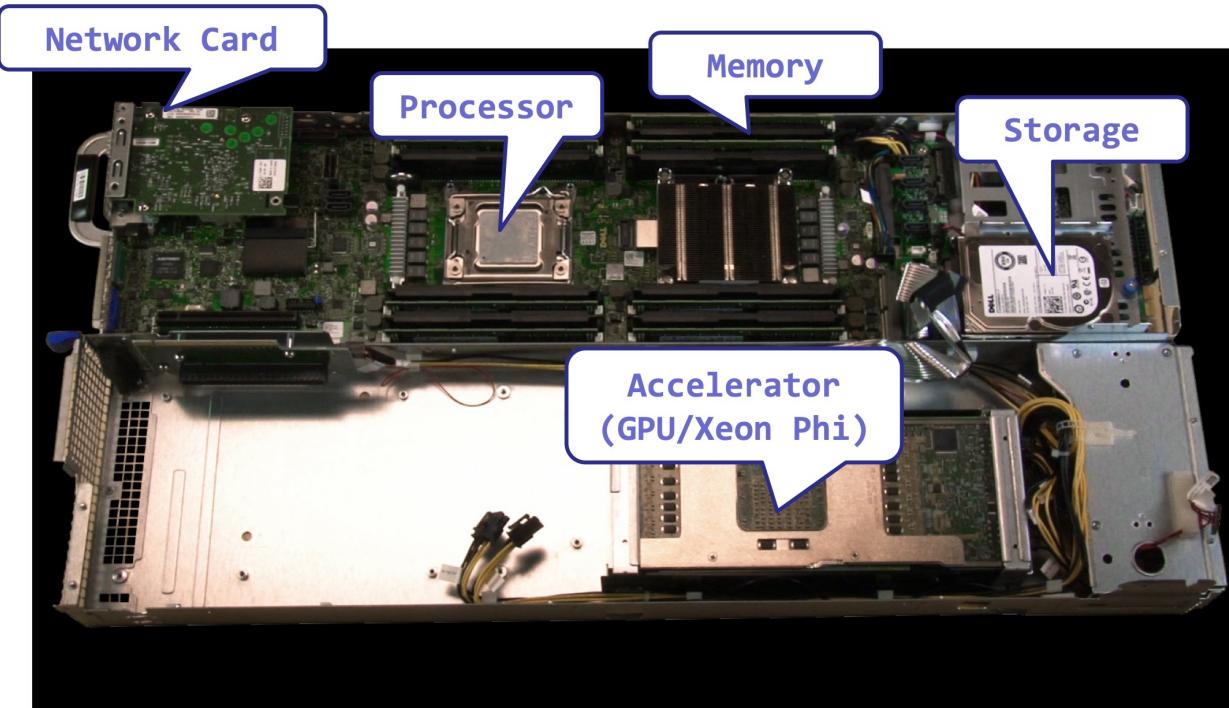
Um típico cluster

- Um rack



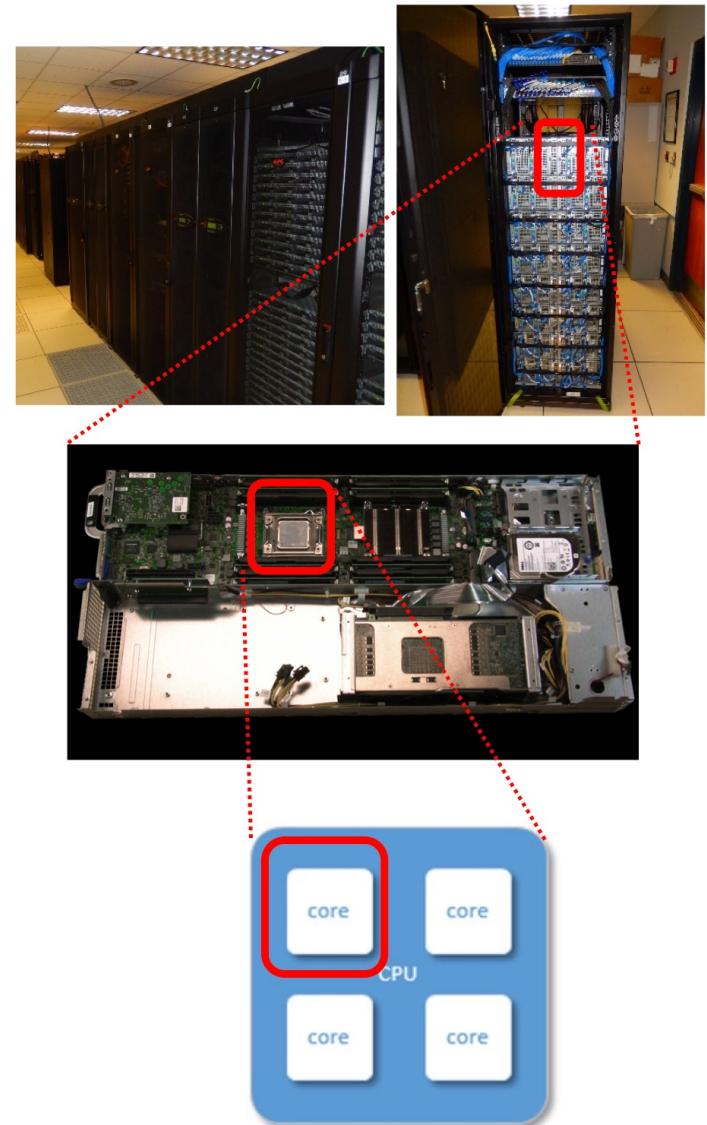
Um típico cluster

- Um node

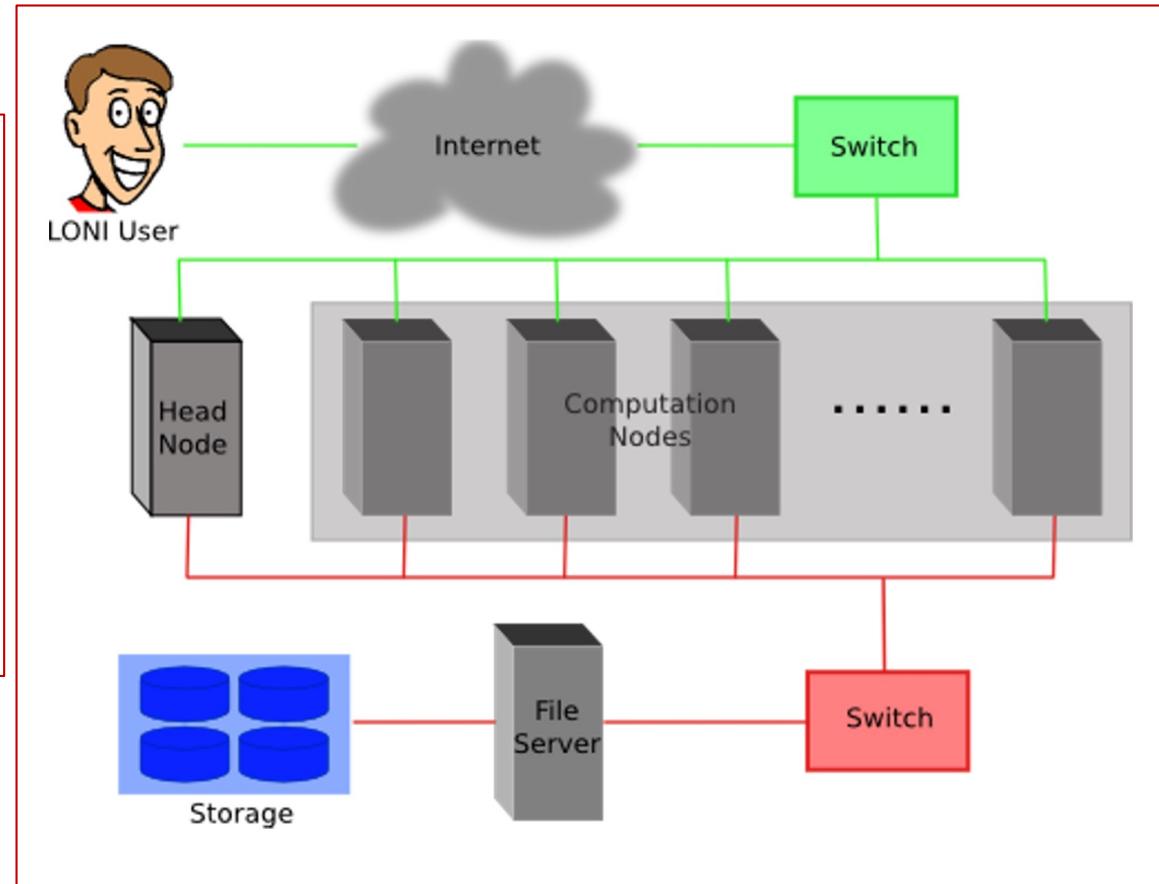
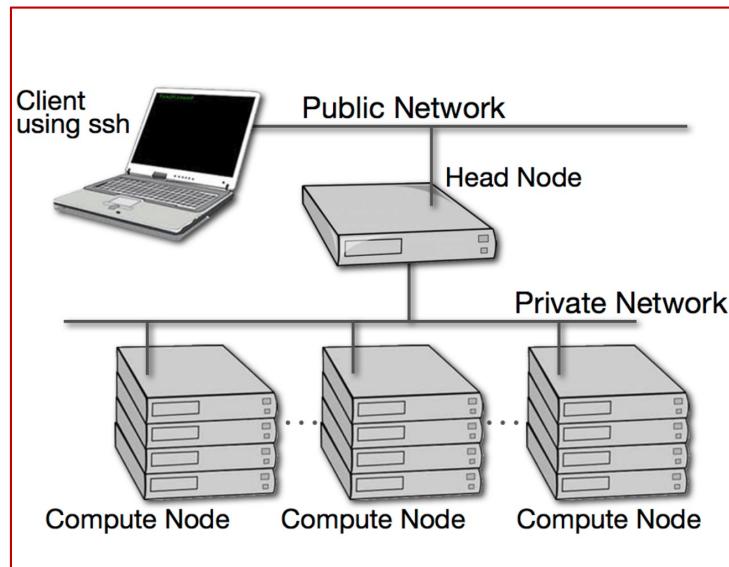


Cluster - Terminologias

Term	Definition
Cluster	A set of connected computer nodes that work together. (E.g., QB2)
Node	A single, named host machine in the cluster. (E.g., qb010)
Core	The basic computation unit in a processor. (E.g. , QB2 has two 10-core processors → 20 cores)
Job	A user's request to use a certain amount of resources for a certain amount of time on cluster for his/her work.



Cluster – Arquitetura convencional



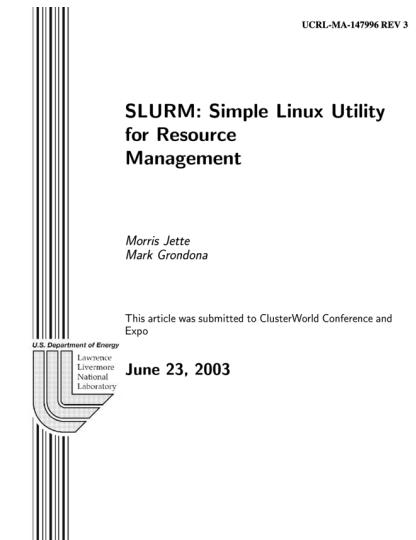
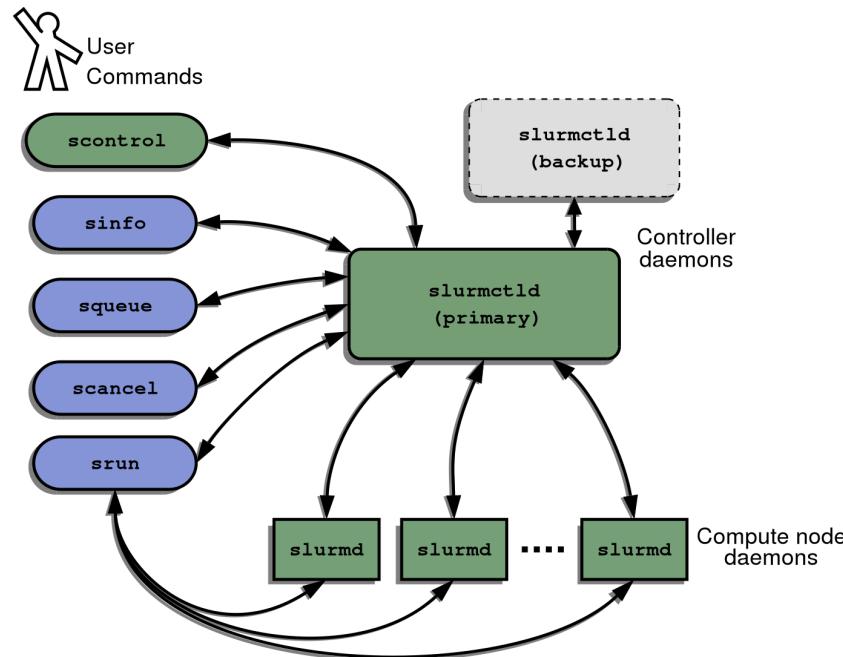
SLURM

- É um acrônimo para “Simple Linux Utility for Resource Management”
- É um sistema de gerenciamento de filas e recursos para clusters Linux
- Sua primeira versão foi em 2003
- Amplamente adotado em ambientes acadêmicos e industriais devido a sua eficiência e simplicidade
- Desenvolvimento é contínuo com contribuições da comunidade open-source
- Atualmente é mantido e atualizado pelo SchedMD
- Aproximadamente 60% dos supercomputadores da lista TOP500 usam SLURM



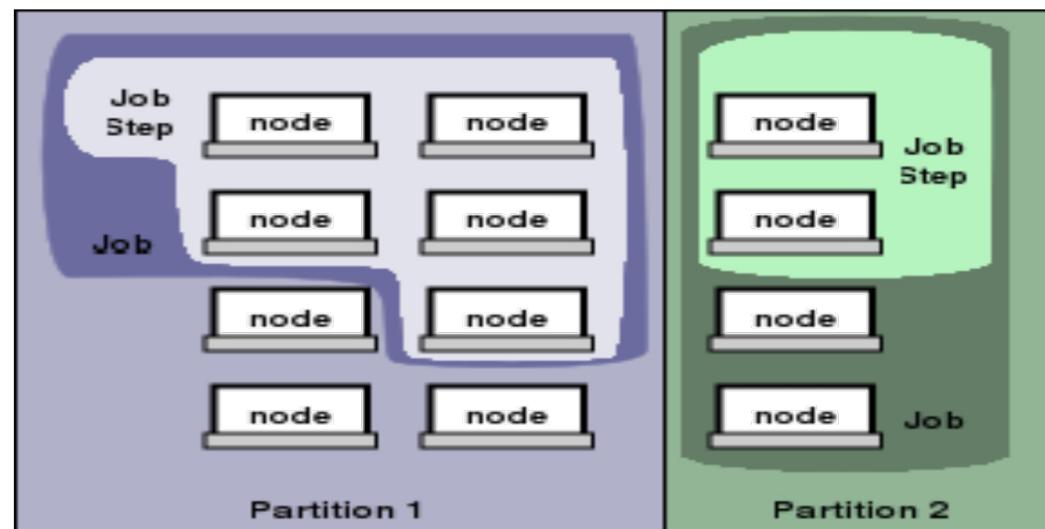
SLURM

- O paper de apresentação do SLURM está [aqui](#)



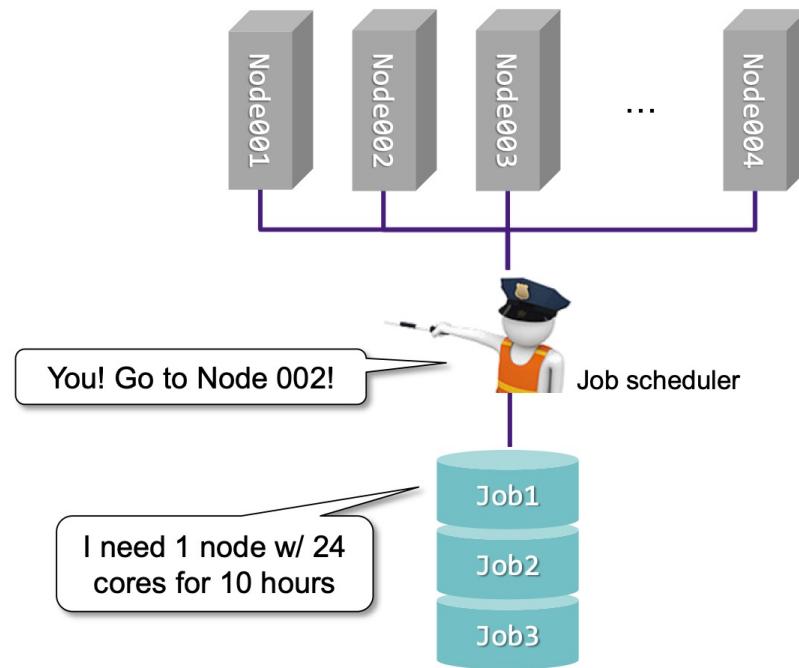
SLURM - terminologia

- **Computing node** Computer used for the execution of programs
- **Partition** Group of nodes into logical sets
- **Job** allocation of resources assigned to a user for some time
- **Step** sets of (possible parallel) tasks with a job



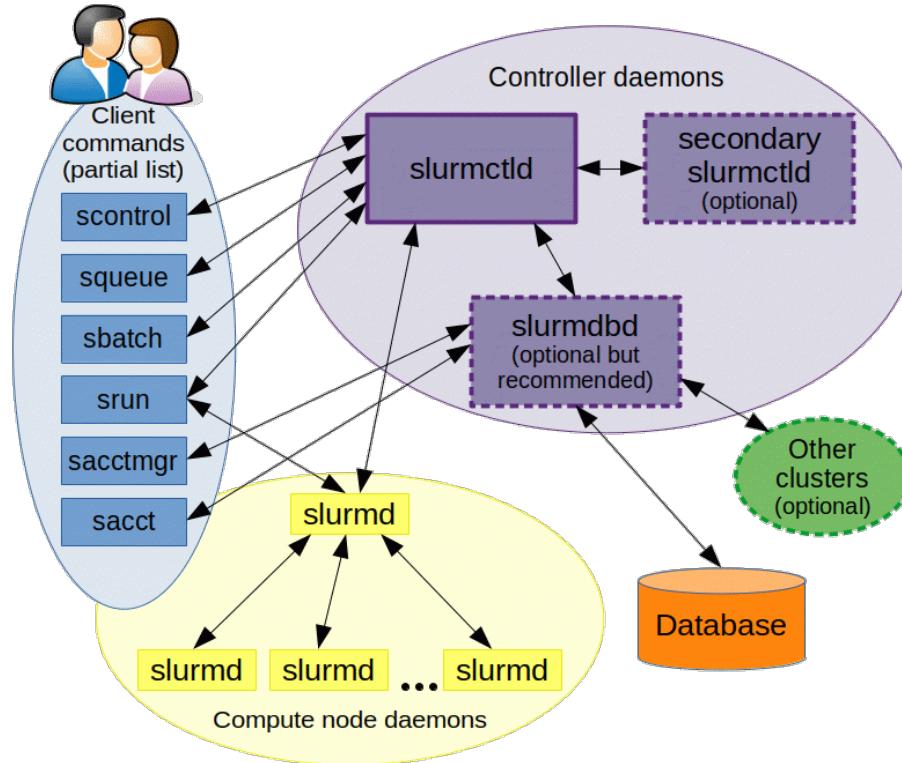
SLURM também é um Scheduler!

- E o que é isso?



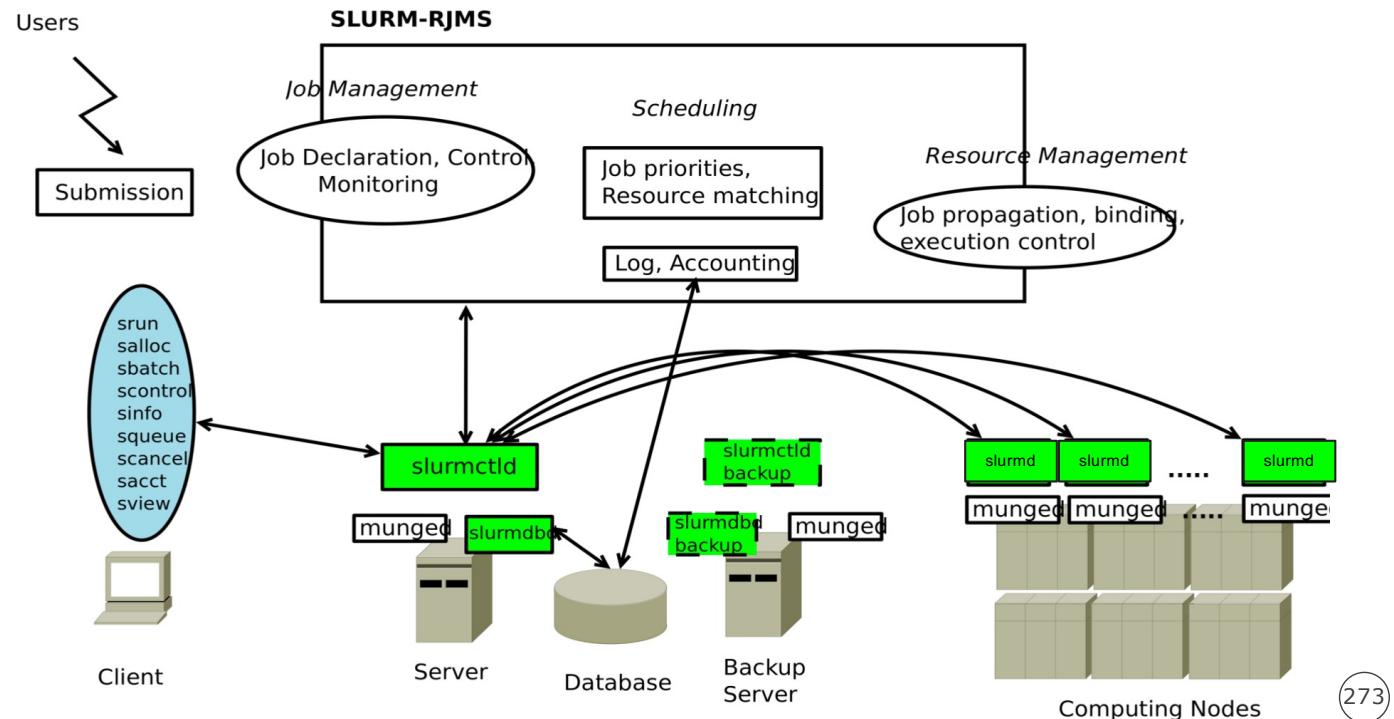
SLURM

- Componentes



SLURM – deamons

- Deamon – programa que executa como um processo em segundo plano



SLURM – Arquivos de configuração

slurm.conf

- Low level configuration
- Management policies
- Scheduling policies
- Allocation policies
- Node definition
- Partition definition

slurmdbd.conf

- Type of persistent storage (DB)
- Location of storage

topology.conf

- Switch hierarchy

gres.conf

- Generic resources details
- Device files

cgroup.conf

- Mount point
- Release agent path
- Cgroup subsystems parameters

	Controller	Compute node
Mandatory	slurm.conf slurmdbd.conf	slurm.conf
Optional	prologs epilogs topology.conf	gres.conf cgroup.conf topology.conf

SLURM – Configuração de nós e partições

- É no arquivo de configuração slurm.conf onde configuramos os nós e partições (filas) do cluster

Node definition

- Characteristics (sockets, cores, threads, memory, features)
- Network addresses

Partition definition

- Set of nodes
- Sharing
- Priority/preemption

```
# Compute Nodes
NodeName=cuzco[1-10] Procs=16 Sockets=2 CoresPerSocket=8 ThreadsPerCore=1 State=UNKNOWN RealMemory=38000
NodeName=cuzco[10-20] Procs=32 Sockets=2 CoresPerSocket=8 ThreadsPerCore=2 State=UNKNOWN RealMemory=46000

# Partitioning
PartitionName=exclusive Nodes=cuzco[1-20] MaxTime=INFINITE State=UP Priority=10 Shared=Exclusive
PartitionName=shared Nodes=berlin[1-20] Default=YES MaxTime=INFINITE State=UP Priority=30
PartitionName=procs16 Nodes=berlin[1-10] MaxTime=INFINITE State=UP Priority=30
PartitionName=procs32 Nodes=berlin[10-20] MaxTime=INFINITE State=UP Priority=30
```

Shared Option

Controls the ability of the partition to execute more than one job on a resource (node, socket, core)

EXCLUSIVE allocates entire node (overrides cons_res ability to allocate cores and sockets to multiple jobs)

NO sharing of any resource.

YES all resources can be shared, unless user specifies –exclusive on srun | salloc | sbatch

Como o SLURM prioriza?



Como o SLURM prioriza?

- Por default, FIFO
- Mas há alguns outros critérios, como:

Age - the length of time a job has been waiting in the queue, eligible to be scheduled

Fairshare - the difference between the portion of the computing resource that has been promised and the amount of resources that has been consumed

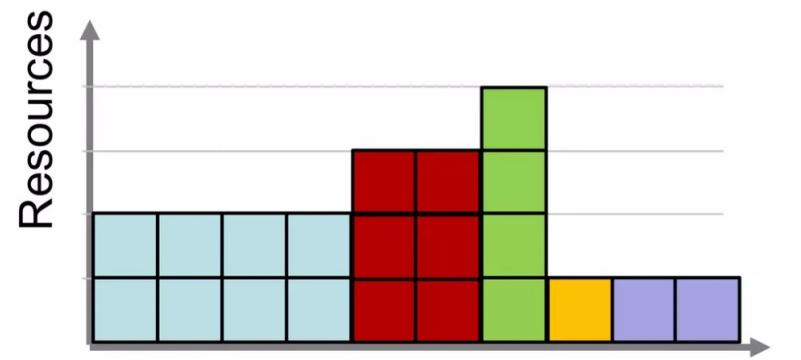
Job size - the number of nodes or CPUs a job is allocated

Partition - a factor associated with each node partition

QOS - A factor associated with each Quality Of Service

Isso tem a ver com algoritmos de escalonamento

FIFO Scheduling



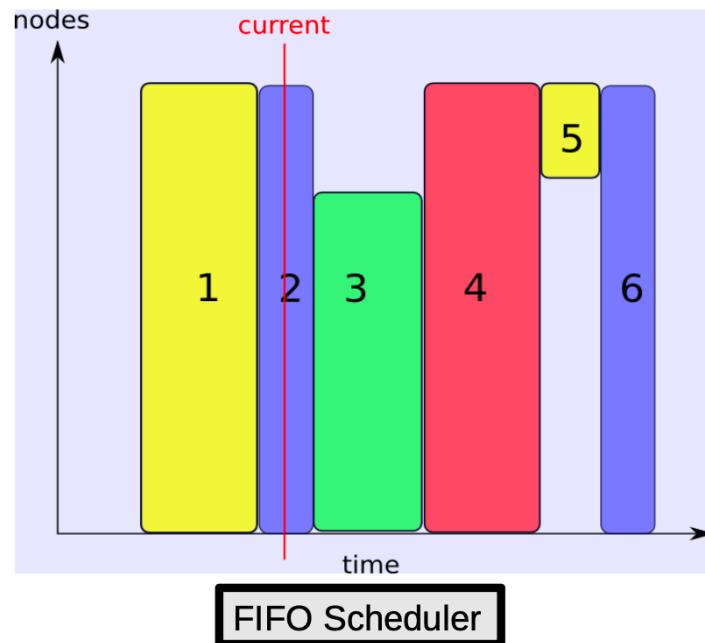
Backfill Scheduling

- Job priority
- Time limit (**Important!**)

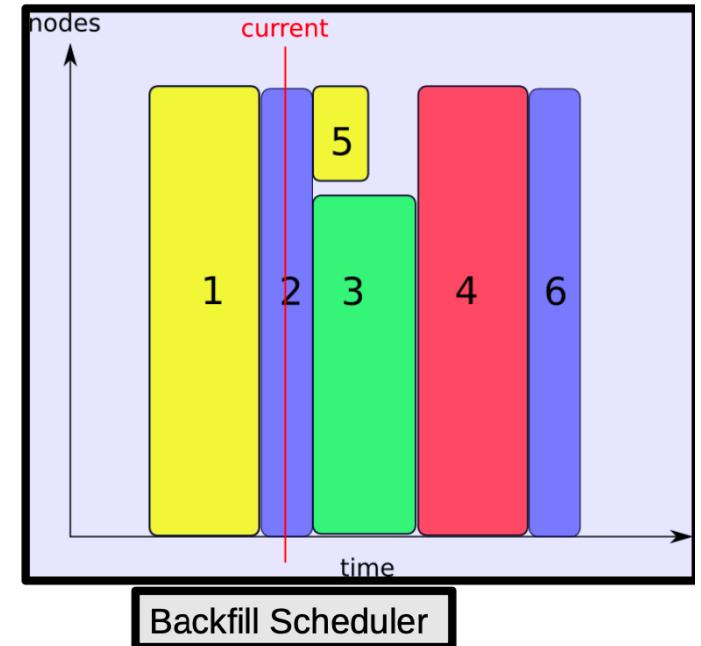


Backfill Scheduler

Holes can be filled if previous jobs order is not changed



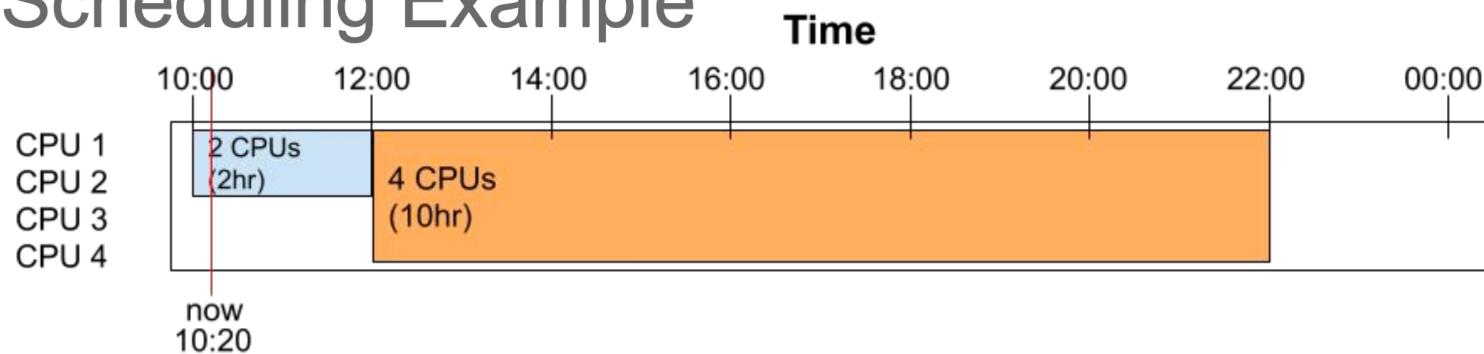
FIFO Scheduler



Backfill Scheduler

Backfill Scheduler

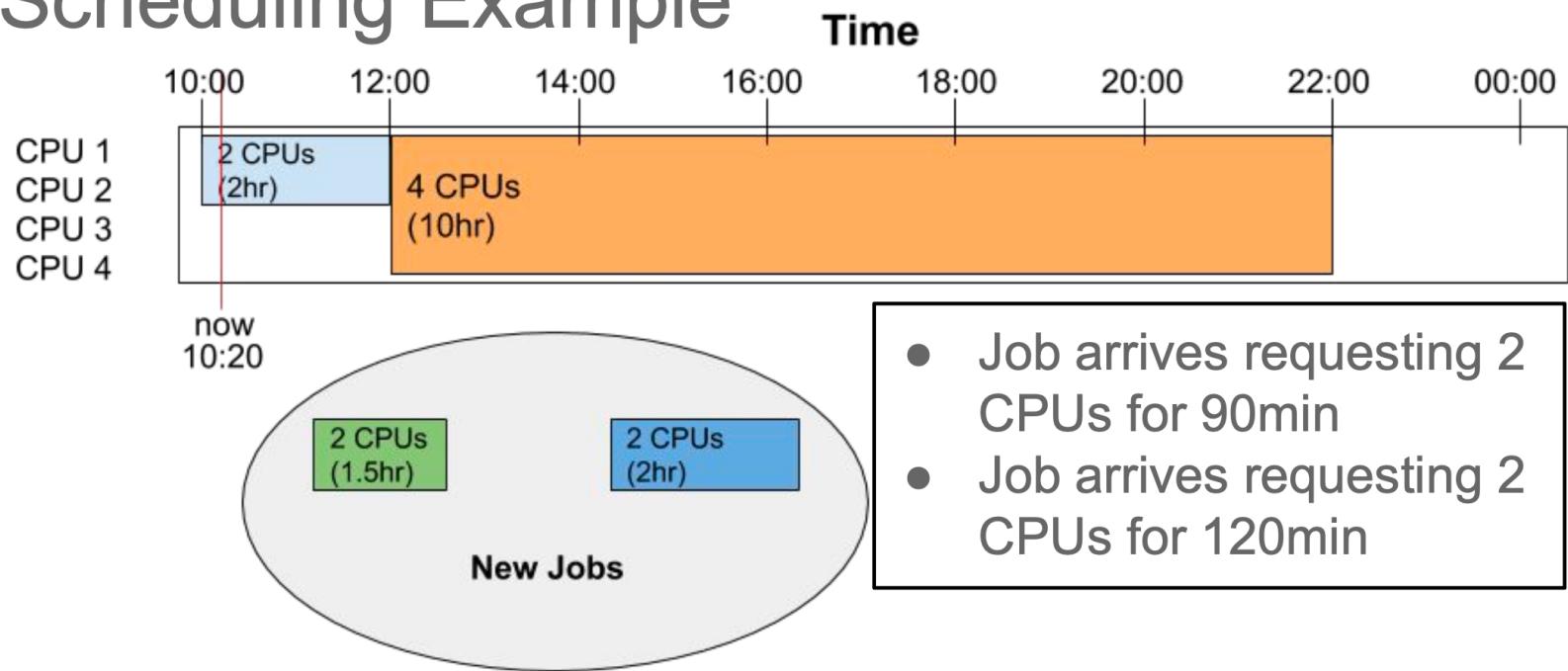
Scheduling Example



- 2 CPU job running until noon
- 4 CPU job starts at noon

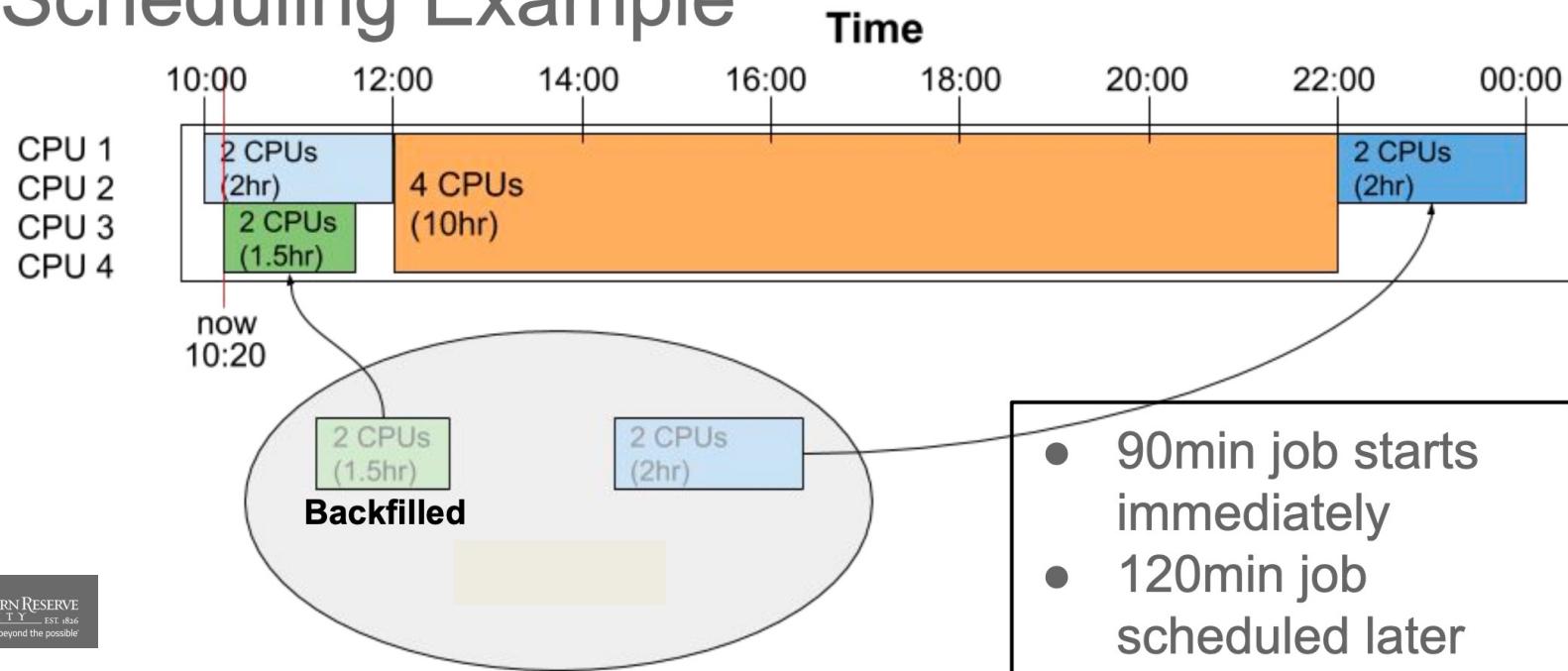
Backfill Scheduler

Scheduling Example



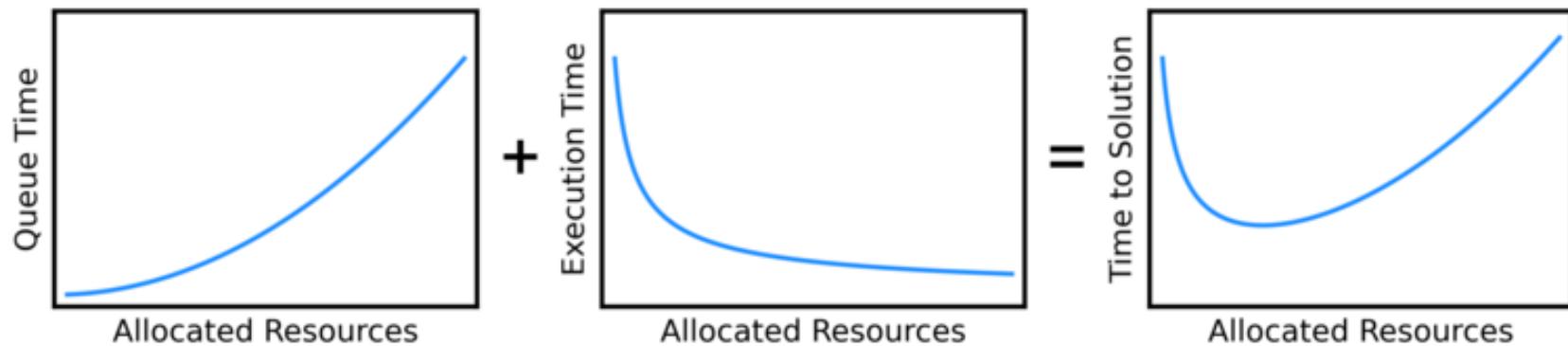
Backfill Scheduler

Scheduling Example

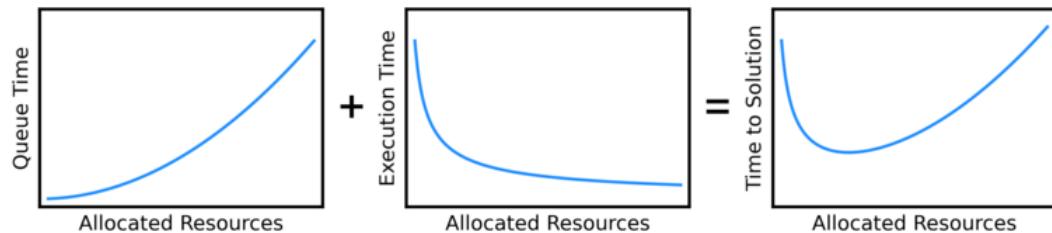


Quanto tempo vai demorar meu job?

- Uma regra simples é escolher o menor conjunto de recursos que proporcione uma aceleração razoável em relação ao baseline



Não é sobre pedir mais, é sobre pedir corretamente!



Ao lado temos um programa OpenMP em um cluster. Conforme aumentamos as threads (cpus-per-task) vamos melhorando o speedup. Mas veja que a partir de 16 threads, nossa eficiência diminui.

ntasks	cpus-per-task	execution time	speed-up ratio	parallel efficiency
1	1	42.0	1.0	100%
1	2	22.0	1.9	95%
1	4	16.0	2.6	66%
1	8	7.8	5.4	67%
1	16	6.5	6.5	40%
1	32	7.1	5.9	18%

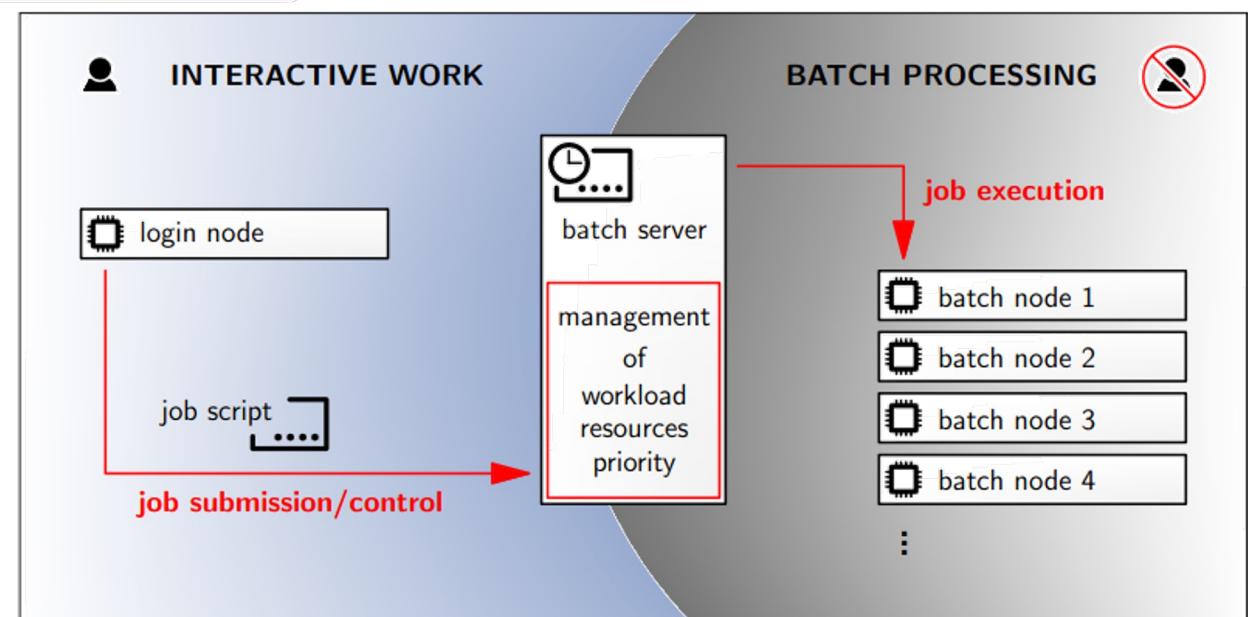
Jobs interativos e Jobs em Batch

1) Interactive job

- Runs **in terminal** (just like using a local machine)
- **Can interact** with the job while running

2) Batch job

- Submit to server and runs **by itself**, until finished or error
- **Cannot interact** with the job while running



Jobs Interativos

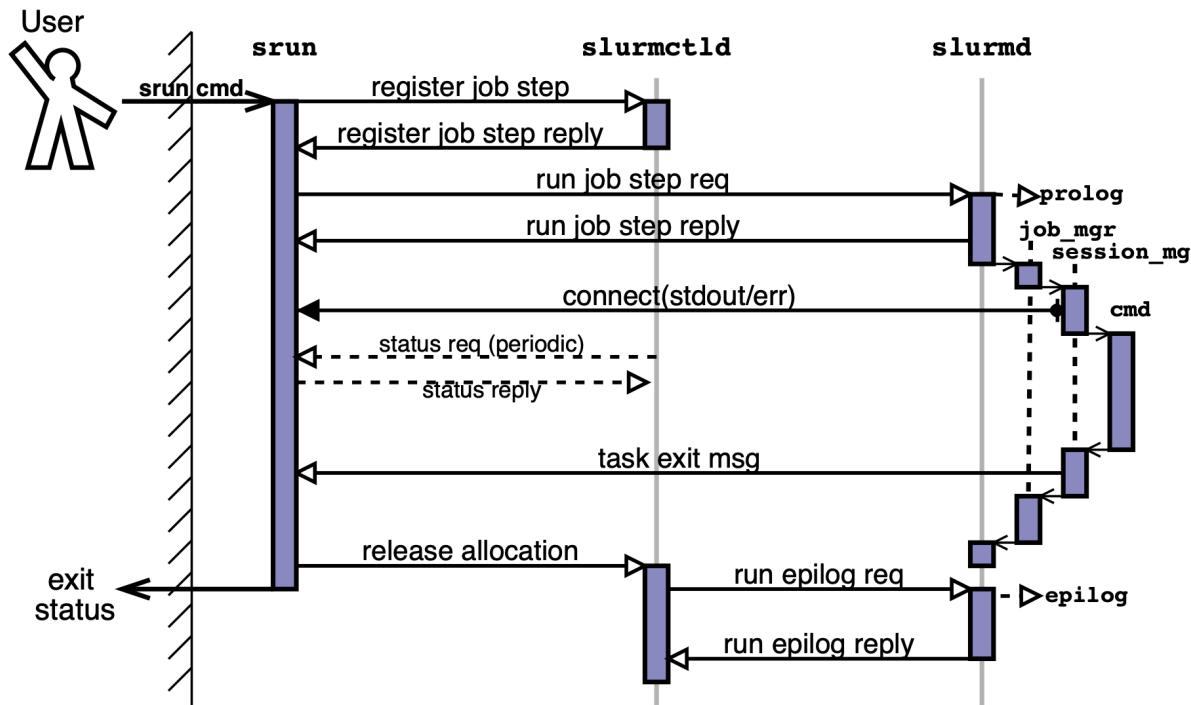


Fig. 5. Interactive job initiation. **srun** simultaneously allocates nodes and a job step from **slurmctld** then sends a run request to all **slurmds** in job. Dashed arrows indicate a periodic request that may or may not occur during the lifetime of the job

Insper

UCRL-MA-147996 REV 3

SLURM: Simple Linux Utility
for Resource
Management

Morris Jette
Mark Grondona

This article was submitted to ClusterWorld Conference and
Expo

June 23, 2003

Approved for public release; further dissemination unlimited

286

Jobs Batch

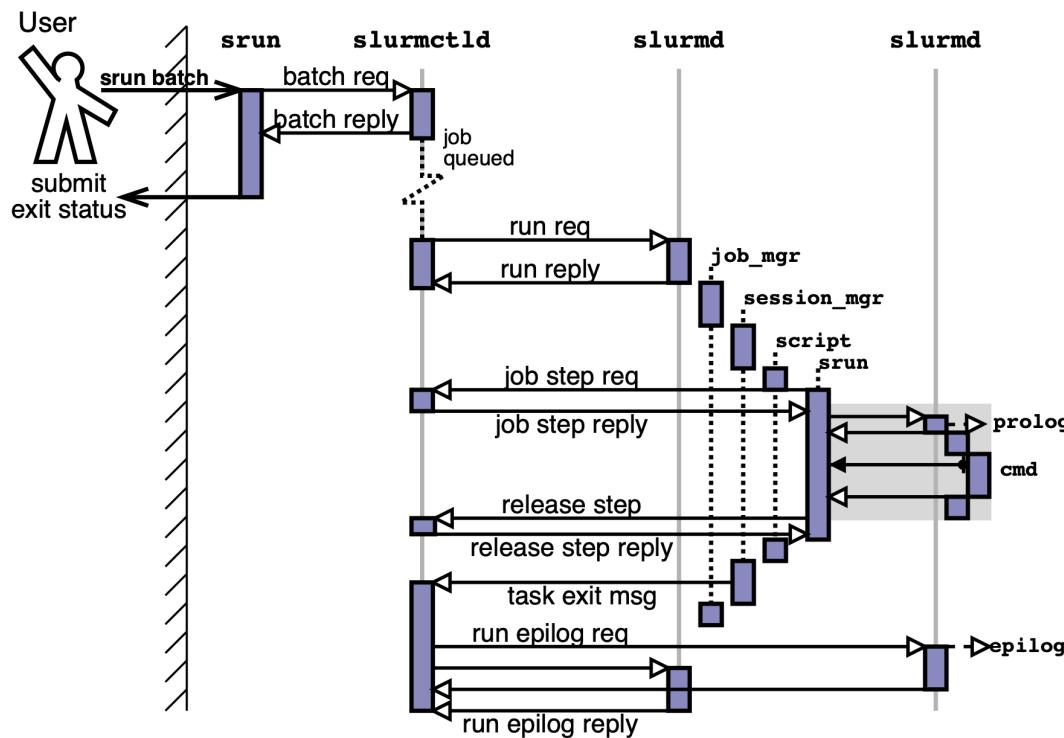


Fig. 6. Queued job initiation. `slurmctld` initiates the user's job as a batch script on one node. Batch script contains an `srun` call that initiates parallel tasks after instantiating job step with controller. The shaded region is a compressed representation and is shown in more detail in the interactive diagram (Figure 5)

Inspire

UCRL-MA-147996 REV 3

SLURM: Simple Linux Utility
for Resource
Management

Morris Jette
Mark Grondona

This article was submitted to ClusterWorld Conference and
Expo

June 23, 2003

Approved for public release; further dissemination unlimited

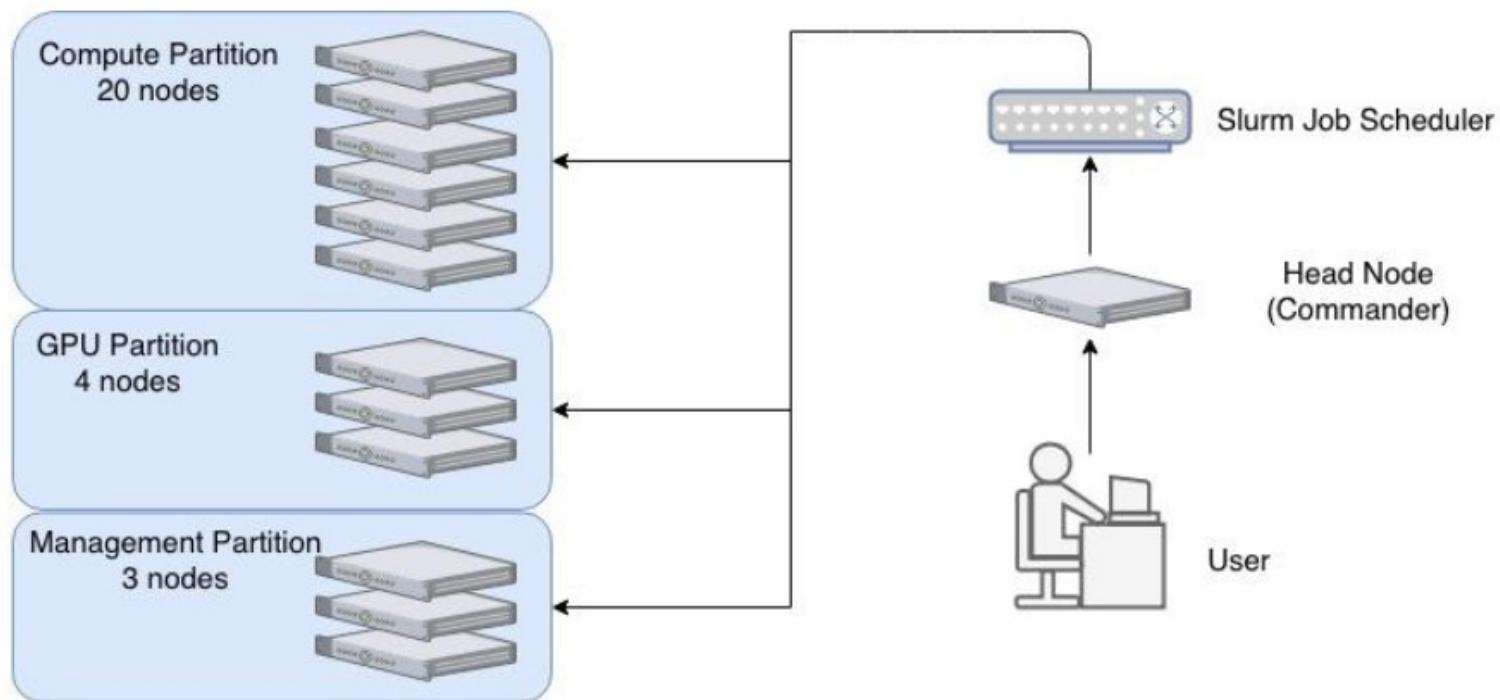
Interativo ou em Batch? R: Use os dois!

Running on HPC: Easy as 1-2-3

1. Plan ahead	2. Test interactively	3. Run remotely
<ul style="list-style-type: none">▪ Organize project directories▪ Install software, transfer data▪ Think about your job requirements<ul style="list-style-type: none">- amount of memory- number of i/o files- size of data	<ul style="list-style-type: none">▪ Use salloc/srun<ul style="list-style-type: none">- test run commands- check paths, results▪ Experiment with resource allocation<ul style="list-style-type: none">- number cores (cpus)- amount of memory▪ Start small!<ul style="list-style-type: none">- <i>then scale</i>	<ul style="list-style-type: none">▪ Create Slurm script<ul style="list-style-type: none">e.g., myjob.slurm▪ Submit job to queue<ul style="list-style-type: none">\$sbatch myjob.slurm▪ Monitor your job<ul style="list-style-type: none">\$squeue --user <uname>▪ Check results<ul style="list-style-type: none">\$less slurm-<jobid>.out

https://chianglab.usc.edu/doc/Intro_to_Slurm.pdf

Partições (FILAS)



Principais comandos do SLURM - Interativo

Useful SLURM Commands		
Command	Summary	Example
srun <code>-parameter job</code>	Obtain a job allocation and execute an application (see Running a Program - SRUN)	srun <code>-N1 -n1 example.py</code>
sbatch <code>-parameter batchscript</code>	Submit a batch script for later execution (see Running a Program - SBATCH)	sbatch <code>-o example.sh</code>
sacct	Display accounting data Use <code>-j</code> jobid to see status of specific job	sacct <code>-j 1576</code>
sinfo	View the status of the cluster's nodes and partitions	sinfo
squeue	Displays information of jobs in queue	squeue

Principais comandos do SLURM - Batch

Common Batch Script Directives		
Directive	Description	Example
--job-name= <code>name</code> or -J <code>name</code>	Custom job name	--job-name= <code>example</code>
--partition= <code>name</code> or -p <code>partition</code>	Partition to run on	--partition= <code>compute</code>
--nodes=# or -N#	Total number of nodes	--nodes=1
--ntasks=# or -n#	Number of "tasks". For use with distributed parallelism. See below.	--ntasks=1
--cpus-per-task=# or -c#	# of CPUs allocated to each task. For use with shared memory parallelism.	--cpus-per-task=1
--ntasks-per-node=#	Number of "tasks" per node. For use with distributed parallelism. See below.	--ntasks-per-node=2
--time=[[DD-]HH:]MM:SS or -t [[DD-]HH:]MM:SS	Maximum walltime of the job in Days-Hours:Minutes:Sec	--time=10:00 10 minutes
--mem=#	Memory requested per node in MB	--mem=1G

Resumo de comandos do SLURM

- Disponível [aqui](#)

 slurm workload manager	
Job Submission	
<code>salloc</code> - Obtain a job allocation.	
<code>shbatch</code> - Submit a batch script for later execution.	
<code>swbatch</code> - Obtain a job allocation (as needed) and execute an application.	
<code>-array <index></code> (e.g. "-array=1-10")	Job array specification (batch command only)
<code>-account <name></code>	Account to be charged for resources used.
<code>-begin <time></code> (e.g. "-begin=18:00:00")	Initiate job after specified time.
<code>-clusters <name></code>	Cluster(s) to run the job. (batch command only)
<code>-constraint <features></code>	Required node features.
<code>-cpus-per-task <count></code>	Number of CPUs required per task.
<code>-dependency <state>:<jobid></code>	Defers job until specified jobs reach specified state.
<code>-error <filename></code>	File in which to store job error messages.
<code>-exclude <names></code>	Specific host names to exclude from job allocation.
<code>-exclusive <user></code>	Allocated nodes can not be shared with other users/users.
<code>-export <name>=<value></code>	Export identified environment variables.
<code>-gres <name>[<count>]</code>	Generic resources required per node.
<code>-input <name></code>	File from which to read job input data.
<code>-job-name <name></code>	Job name.
<code>-label</code>	Prepend task ID to output. (from command only)
<code>-licenses <name>[<count>]</code>	License resources required for entire job.
Accounting	
<code>sacctm</code> - View and modify account information.	
<code>Options:</code>	
<code>-mem <MB></code>	Memory required per node.
<code>-mem-per-cpu <MB></code>	Memory required per allocated CPU.
<code>-N<n>minnodes[<maxnodes>]</code>	Node count required for the job.
<code>-o<count></code>	Number of tasks to be launched.
<code>-nodelist <names></code>	Specify host names to include in job allocation.
<code>-output <name></code>	File in which to store job output.
<code>-partition <names></code>	Partition/queue in which to run the job.
<code>-qos <name></code>	Quality Of Service.
<code>-signal <[B]>num[<@time>]</code>	Signal job at approaching time limit.
<code>-time <time></code>	Wall clock time limit.
<code>-wrap <command> <string></code>	Wrap specified command in a simple "sh" shell. (batch command only)
Job Management	
<code>scancel</code> - Transfer files to a job's compute nodes.	
<code>scast [options] SOURCE DESTINATION</code>	
<code>-force</code>	Replace previously existing file.
<code>-preserve</code>	Preserve modification times, access times, and access permissions.
<code>scancel</code> - Signal jobs, job arrays, and/or job steps.	
<code>-account <name></code>	Operate only on jobs charging the specified account.
<code>-name <name></code>	Operate only on jobs with specified name.
<code>-partition <names></code>	Comma separated list of partitions to select jobs and job steps from.
<code>-state <state>[<list>]</code>	Display jobs with specified states.
<code>-starttime <time></code>	Start of reporting period.
<code>sinfo</code> - View information about nodes and partitions.	
<code>-all</code>	Display information about all partitions.
<code>-dead</code>	If set, only report state information for non responding (dead) nodes.
Environment Variables	
<code>SLURM_ARRAY_TASK_ID</code>	Set to the task ID if part of a job array.
<code>SLURM_CLUSTER_NAME</code>	Name of the cluster executing the job.
<code>SLURM_CPUS_PER_TASK</code>	Number of CPUs requested per task.
<code>SLURM_JOB_ACCOUNT</code>	Account name.
<code>SLURM_JOB_ID</code>	Job ID.
<code>SLURM_JOB_NAME</code>	Job Name.
<code>SLURM_JOB_NODELIST</code>	Names of nodes allocated to job.
<code>SLURM_JOB_NUM_NODES</code>	Number of nodes allocated to job.
<code>SLURM_JOB_PARTITION</code>	Partition/queue running the job.
<code>SLURM_JOB_UID</code>	User ID of the job's owner.
<code>SLURM_JOB_USER</code>	User name of the job's owner.
<code>SLURM_RESTART_COUNT</code>	Number of times job has restarted.
<code>SLURM_PROCID</code>	Task ID (MPI rank).
<code>SLURM_STEP_ID</code>	Job step ID.
<code>SLURM_STEP_NUM_TASKS</code>	Task count (number of MPI ranks).
Daemons	
<code>slurmd</code>	Executes on cluster's "head" node to manage workload.
<code>slurm</code>	Executes on each compute node to locally manage resources.
<code>slurmdbd</code>	Manages database of resources limits, licenses, and archives accounting records.

Atividade

- Em nosso cluster:
 - 1 – Aprenderemos a verificar o status do cluster (sinfo, scontrol)
 - 2 – Criaremos uma nova partição (fila) para permitir que Jobs executem no máximo por 5 minutos e usando apenas 1 CPU (chamaremos essa fila de express) [modificaremos o /etc/slurm/slurm.conf no master]
 - 3 – Ampliaremos a memória disponível ao SLURM nos nós de computação
 - 4 - Submeteremos um job interativo (srun)
 - 5 – Submeteremos alguns Jobs em batch (sbatch) e monitorar a fila (squeue), além de realizar alguns comandos administrativos (scancel, shold, sacct)
 - 6 – Monitoraremos o desempenho no Ganglia

Parte prática



Inicialização

1. Inicialize as máquinas virtuais (comece pela smshost)
2. Configure o redirect da porta 8080 da sua máquina física para a 80 da sms-host
3. Vá no diretório onde há os arquivos do vagrant, e execute **vagrant ssh**. Com isso, você deve logar na máquina sms

Port Forwarding Rules

Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
ssh	TCP	127.0.0.1	2229	22	22
web	TCP	127.0.0.1	8080	80	80

```
cluster@cluster-NUC7i5DNHE:~/VagrantCluster$ vagrant ssh
Last login: Thu Nov  2 20:07:02 2023 from 10.0.2.2
This system is built by the Bento project by Chef Software
More information can be found at https://github.com/chef/bento
[vagrant@sms-host ~]$ sudo su
[root@sms-host vagrant]#
```

GIT Clone

- Alguns de nossos arquivos estão hospedados no git.
- Execute um clone na máquina master

```
git clone https://github.com/andrefmb/slurm_pratica.git
```

Verificar deamons

- Veja se o deamon **slurmctld** está executando

```
[root@sms-host vagrant]# systemctl status slurmctld
● slurmctld.service - Slurm controller daemon
  Loaded: loaded (/usr/lib/systemd/system/slurmctld.service; enabled; vendor preset: disabled)
  Active: active (running) since Sáb 2023-11-04 11:50:37 UTC; 11min ago
    Process: 1246 ExecStart=/usr/sbin/slurmctld $SLURMCTLD_OPTIONS (code=exited, status=0/SUCCESS)
   Main PID: 1305 (slurmctld)
     CGroup: /system.slice/slurmctld.service
             └─1305 /usr/sbin/slurmctld
```

Verificar deamons

- Veja agora, com o auxílio do **pdsh**, se o deamon **slurmd** está executando nos nós.

```
[root@sms-host vagrant]# pdsh -w compute0[0-1] systemctl status slurmd
compute01: ● slurmd.service - Slurm node daemon
compute01:   Loaded: loaded (/usr/lib/systemd/system/slurmd.service; disabled; vendor preset: disabled)
compute01:   Active: inactive (dead)
pdsh@sms-host: compute01: ssh exited with exit code 3
compute00: ● slurmd.service - Slurm node daemon
compute00:   Loaded: loaded (/usr/lib/systemd/system/slurmd.service; disabled; vendor preset: disabled)
compute00:   Active: inactive (dead)
pdsh@sms-host: compute00: ssh exited with exit code 3
[root@sms-host vagrant]#
```

- Nesse caso, não está. Então devemos executar o comando abaixo

```
[root@sms-host vagrant]# pdsh -w compute0[0-1] systemctl start slurmd
```

Verificar deamons

- Execute novamente para ver o resultado

```
[root@sms-host vagrant]# pdsh -w compute0[0-1] systemctl status slurmd
compute00: ● slurmd.service - Slurm node daemon
compute00:   Loaded: loaded (/usr/lib/systemd/system/slurmd.service; disabled; vendor preset: disabled)
compute00:   Active: active (running) since Sáb 2023-11-04 11:59:16 UTC; 3s ago
compute00:     Process: 1787 ExecStart=/usr/sbin/slurmd $SLURMD_OPTIONS (code=exited, status=0/SUCCESS)
compute00:    Main PID: 1789 (slurmd)
compute00:      Tasks: 1
compute00:        Memory: 1.0M
compute00:        CGroup: /system.slice/slurmd.service
compute00:                  └─1789 /usr/sbin/slurmd
compute00:
compute00: Nov 04 11:59:16 compute00.hpcnet systemd[1]: Starting Slurm node daemon...
compute00: Nov 04 11:59:16 compute00.hpcnet systemd[1]: Started Slurm node daemon.
compute01: ● slurmd.service - Slurm node daemon
compute01:   Loaded: loaded (/usr/lib/systemd/system/slurmd.service; disabled; vendor preset: disabled)
compute01:   Active: active (running) since Sáb 2023-11-04 11:59:17 UTC; 3s ago
compute01:     Process: 1808 ExecStart=/usr/sbin/slurmd $SLURMD_OPTIONS (code=exited, status=0/SUCCESS)
compute01:    Main PID: 1810 (slurmd)
compute01:      Tasks: 1
compute01:        Memory: 1.0M
compute01:        CGroup: /system.slice/slurmd.service
compute01:                  └─1810 /usr/sbin/slurmd
compute01:
compute01: Nov 04 11:59:16 compute01.hpcnet systemd[1]: Starting Slurm node daemon...
compute01: Nov 04 11:59:17 compute01.hpcnet systemd[1]: Started Slurm node daemon.
[root@sms-host vagrant]# █
```

Verifique o status do cluster

- Verifique o status do cluster com `sinfo`

```
[root@sms-host vagrant]# sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
normal*      up    1-00:00:00      2  down  compute[00-01]
```

- Veja que o nó está com o STATE = DOWN. Isso provavelmente se dá porque o controlador do SLURM não atualizou o status do nó. Garanta que como root você faz ssh aos nós (compute00 e compute01). Se isso funcionar, vamos executar o comando `scontrol` para atualizar o status dos nós:

```
[root@sms-host vagrant]# scontrol update nodename=compute00 state=RESUME
[root@sms-host vagrant]# scontrol update nodename=compute01 state=RESUME
```

```
[root@sms-host vagrant]# sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
normal*      up    1-00:00:00      2  idle  compute[00-01]
```

Primeiro job interativo

- Vamos submeter um job interativo (**srun**), para executar por 1 minuto, e vamos ver qual nó o SLURM vai alocar. Vamos aguardar o SLURM finalizar nosso job pelo walltime. Aproveite para abrir uma nova aba do terminal, para monitorar a fila (**squeue**)

```
[root@sms-host vagrant]# su - test
Last login: Sáb Nov  4 11:53:54 UTC 2023 on pts/0
[test@sms-host ~]$ srun -N1 -t00:01:00 --pty /bin/bash
[test@compute00 ~]$
```

```
[test@sms-host ~]$ squeue
      JOBID PARTITION      NAME      USER ST      TIME  NODES NODELIST(REASON)
          39    normal      bash     test   R      0:10      1 compute00
```

Resumo de nosso job

- Vamos ver um resumo de como foi nosso job. Para isso, usamos o comando `sacct <job_id>`

```
[test@sms-host ~]$ sacct -j 39
      JobID      JobName  Partition      Account AllocCPUS      State ExitCode
-----  -----  -----  -----  -----
39          bash    normal    (null)        0  TIMEOUT      0:0
```

podemos perceber que o status de nosso job é TIMEOUT, isto é, o SLURM finalizou nosso job em função do walltime.

Mudando nosso cluster

- Resgatando o **sinfo**, vamos ver que temos apenas uma fila normal

```
[root@sms-host vagrant]# sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
normal*      up   1-00:00:00      2  idle  compute[00-01]
              5-00      0 idle  compute[00-01]
```

- Vamos editar o arquivo **/etc/slurm/slurm.conf** no master, para aumentar a memória de cada nó e também criar uma nova fila

```
## COMPUTE NODES
# OpenHPC default configuration
TaskPlugin=task/affinity
PropagateResourceLimitsExcept=MEMLOCK
AccountingStorageType=accounting_storage/filetxt
Epilog=/etc/slurm/slurm.epilog.clean
NodeName=compute0[0-1] Sockets=1 CoresPerSocket=2 RealMemory=3000 ThreadsPerCore=1 State=UNKNOWN
PartitionName=normal Nodes=compute0[0-1] Default=YES MaxMemPerNode=3000 MaxTime=24:00:00 State=UP
PartitionName=express Nodes=compute0[0-1] Default=NO MaxMemPerNode=1000 MaxTime=00:05:00 State=UP MaxNodes=1 MaxCPUsPerNode=1
ReturnToService=1
HealthCheckProgram=/usr/sbin/nhc
HealthCheckInterval=300
```

Mudando nosso cluster

- Reinicie o **slurmctld** e veja se a nova fila é exibida
- Execute o comando **scontrol show nodes** para ver detalhes dos nós. Note as características do nó, em especial RealMemory, e veja se as duas partções (normal, express) estão alocadas para eles

```
[root@sms-host vagrant]# systemctl restart slurmctld
[root@sms-host vagrant]# sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
normal*      up   1-00:00:00      2  idle  compute[00-01]
express       up      5:00      2  idle  compute[00-01]
[root@sms-host vagrant]#
```

```
[root@sms-host vagrant]# scontrol show nodes
NodeName=compute00 Arch=x86_64 CoresPerSocket=2
CPUAlloc=0 CPUTot=2 CPULoad=0.01
AvailableFeatures=(null)
ActiveFeatures=(null)
Gres=(null)
NodeAddr=compute00 NodeHostName=compute00 Version=18.08
OS=Linux 3.10.0-1062.el7.x86_64 #1 SMP Wed Aug 7 18:08:02 UTC 2019
RealMemory=3000 AllocMem=0 FreeMem=1593 Sockets=1 Boards=1
State=IDLE ThreadsPerCore=1 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
Partitions=normal,express
BootTime=2023-11-04T11:51:09 SlurmStartTime=2023-11-04T11:59:17
CfgTRES=cpu=2,mem=3000M,billing=2
AllocTRES=
CapWatts=n/a
CurrentWatts=0 LowestJoules=0 ConsumedJoules=0
ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
```

```
NodeName=compute01 Arch=x86_64 CoresPerSocket=2
CPUAlloc=0 CPUTot=2 CPULoad=0.01
AvailableFeatures=(null)
ActiveFeatures=(null)
Gres=(null)
NodeAddr=compute01 NodeHostName=compute01 Version=18.08
OS=Linux 3.10.0-1062.el7.x86_64 #1 SMP Wed Aug 7 18:08:02 UTC 2019
RealMemory=3000 AllocMem=0 FreeMem=1591 Sockets=1 Boards=1
State=IDLE ThreadsPerCore=1 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
Partitions=normal,express
BootTime=2023-11-04T11:51:05 SlurmStartTime=2023-11-04T11:59:17
CfgTRES=cpu=2,mem=3000M,billing=2
AllocTRES=
CapWatts=n/a
CurrentWatts=0 LowestJoules=0 ConsumedJoules=0
ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
```

Execução de programas

- Hello, World
- Vamos submete-lo interativamente:

```
[test@sms-host ~]$ srun -N1 -p express --pty ./hello_world
Hostname:compute00.hpcnet
RAM Total:3006 MB
RAM Livre:1588 MB
System uptime: 2257 segundos
[test@sms-host ~]$
```

```
#include<iostream>
#include<unistd.h>
#include <sys/sysinfo.h>
using namespace std;
int main(){
    char hostname[256];
    gethostname(hostname, sizeof(hostname));
    struct sysinfo info;
    sysinfo(&info);

    cout << "Hostname:" << hostname << endl;
    cout << "RAM Total:" << info.totalram * info.mem_unit / (1024 * 1024) << " MB" << endl;
    cout << "RAM Livre:" << info.freeram * info.mem_unit / (1024 * 1024) << " MB" << endl;
    cout << "System uptime: " << info.uptime << " segundos" << endl;

    return 0;
}
```

Execução de programas

- Hello, World – agora em batch
- Vamos agora submeter em batch. Nesse caso, vamos criar um [hello_world.slurm](#)

```
[test@sms-host ~]$ squeue
      JOBID PARTITION      NAME     USER ST      TIME  NODES NODELIST(REASON)
        41    express hello_wo   test  PD      0:00      1 (MaxMemPerLimit)
[test@sms-host ~]$
```

Note que nosso job está em PD (pending) e a razão é MaxMemPerLimit. Acontece que nós definimos 1000MB para um job na express e estamos pedindo 1GB. Vamos cancelar e solicitar 500MB

```
#!/bin/bash
#SBATCH --job-name=hello_world
#SBATCH --nodes=1
#SBATCH --partition=express
#SBATCH --mem=1G

echo Output do job $SLURM_JOB_ID
./hello_world
```

Execução de programas

- Para cancelar use `scancel <jobid>`
- Modifique o script para 500M

```
[test@sms-host ~]$ sbatch hello_world.slurm
Submitted batch job 42
[100%]
```

```
[test@sms-host ~]$ sacct -j 42
JobID      JobName      Partition      Account      AllocCPUS      State ExitCode
-----      -----      -----      -----      -----      -----      -----
42          hello_wor+    express        (null)        0      COMPLETED      0:0
```

```
[test@sms-host ~]$ scancel 41
[test@sms-host ~]$ squeue
JOBID PARTITION      NAME      USER ST      TIME NODES NODELIST(REASON)
[test@sms-host ~]$ sacct -j 41
JobID      JobName      Partition      Account      AllocCPUS      State ExitCode
-----      -----      -----      -----      -----      -----      -----
41          hello_wor+    express        (null)        0      CANCELLED      0:0
```

```
#!/bin/bash
#SBATCH --job-name=hello_world
#SBATCH --nodes=1
#SBATCH --partition=express
#SBATCH --mem=500M

echo Output do job $SLURM_JOB_ID
./hello_world
```

```
[test@sms-host ~]$ cat slurm-42.out
Output do job 42
Hostname:compute00.hpcnet
RAM Total:3006 MB
RAM Livre:1587 MB
System uptime: 2832 segundos
```

Execução de programas

```
#!/bin/bash
#SBATCH --job-name=loop_serial
#SBATCH --nodes=1
#SBATCH --partition=express
#SBATCH --mem=500M

echo Output do job $SLURM_JOB_ID
./loop_serial
```

```
#include<iostream>
#include<cmath>
#include<vector>
#include<fstream>
using namespace std;

int main(){
    const int size = 50000000;
    vector<double> results(size);
    double sum = 0.0;

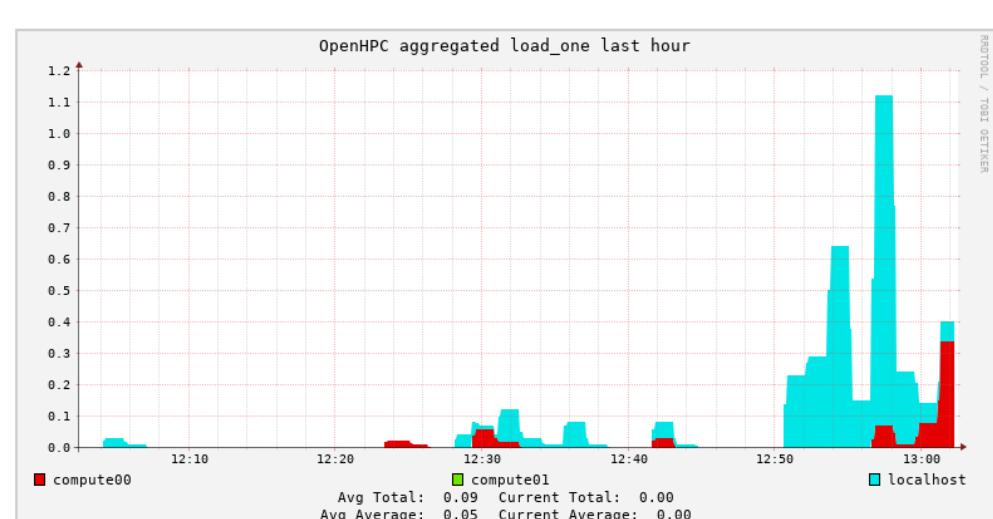
    for(int i = 0; i < size; ++i) {
        results[i] = sqrt(static_cast<double>(i));
        sum += results[i];
    }
    ofstream outfile("simulation_results.txt");
    for(auto &value: results){
        outfile << value << "\n";
    }
    outfile.close();
    cout << "Simulacao completa. Soma das raizes = " << sum << endl;
    return 0;
}
```

```
[test@sms-host ~]$ sbatch loop_serial.slurm
Submitted batch job 44
[test@sms-host ~]$ squeue
      JOBID PARTITION      NAME      USER ST      TIME  NODES NODELIST(REASON)
          44   express  loop_ser     test  R      0:05      1 compute00
[test@sms-host ~]$
```

Execução de programas

- Não esqueça de monitorar via Ganglia

```
[test@sms-host ~]$ scontrol show job 44
JobId=44 JobName=loop_serial
UserId=test(1001) GroupId=test(1001) MCS_label=N/A
Priority=4294901755 Nice=0 Account=(null) QOS=(null)
JobState=COMPLETED Reason=None Dependency=(null)
Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
RunTime=00:01:05 TimeLimit=00:05:00 TimeMin=N/A
SubmitTime=2023-11-04T12:59:22 EligibleTime=2023-11-04T12:59:22
AccrueTime=2023-11-04T12:59:22
StartTime=2023-11-04T12:59:22 EndTime=2023-11-04T13:00:27 Deadline=N/A
PreemptTime=None SuspendTime=None SecsPreSuspend=0
LastSchedEval=2023-11-04T12:59:22
Partition=express AllocNode:Sid=sms-host:1804
ReqNodeList=(null) ExcNodeList=(null)
NodeList=compute00
BatchHost=compute00
NumNodes=1 NumCPUs=2 NumTasks=0 CPUS/Task=1 ReqB:S:C:T=0:0:0:0
TRES=cpu=2,mem=500M,node=1,billing=2
Socks/Node=* NtasksPerN:B:S:C=0:0:0:0 CoreSpec=*
MinCPUsNode=1 MinMemoryNode=500M MinTmpDiskNode=0
Features=(null) DelayBoot=00:00:00
OverSubscribe=NO Contiguous=0 Licenses=(null) Network=(null)
Command=/home/test/loop_serial.slurm
WorkDir=/home/test
StdErr=/home/test/slurm-44.out
StdIn=/dev/null
StdOut=/home/test/slurm-44.out
Power=
```



Execução de programas

- Loop – agora com OpenMP

```
[test@sms-host ~]$ scontrol show job 47
JobId=47 JobName=loop_serial
UserId=test(1001) GroupId=test(1001) MCS_label=N/A
Priority=4294901752 Nice=0 Account=(null) QoS=(null)
JobState=COMPLETED Reason=None Dependency=(null)
Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
RunTime=00:00:59 TimeLimit=00:05:00 TimeMin=N/A
SubmitTime=2023-11-04T13:08:40 EligibleTime=2023-11-04T13:08:40
AccrueTime=2023-11-04T13:08:40
StartTime=2023-11-04T13:08:40 EndTime=2023-11-04T13:09:39 Deadline=N/A
PreemptTime=None SuspendTime=None SecsPreSuspend=0
LastSchedEval=2023-11-04T13:08:40
Partition=express AllocNode:Sid=sms-host:1804
ReqNodeList=(null) ExcNodeList=(null)
NodeList=compute00
BatchHost=compute00
NumNodes=1 NumCPUs=2 NumTasks=1 CPUs/Task=2 ReqB:S:C:T=0:0:2:*
TRES=cpu=2,mem=500M,node=1,billing=2
Socks/Node=* NtasksPerN:B:S:C=0:0:2: CoreSpec=*
MinCPUsNode=2 MinMemoryNode=500M MinTmpDiskNode=0
Features=(null) DelayBoot=00:00:00
OverSubscribe=NO Contiguous=0 Licenses=(null) Network=(null)
Command=/home/test/loop_omp.slurm
WorkDir=/home/test
StdErr=/home/test/slurm-47.out
StdIn=/dev/null
StdOut=/home/test/slurm-47.out
Power=
```

```
#!/bin/bash
#SBATCH --job-name=loop_OMP
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=2
#SBATCH --partition=express
#SBATCH --mem=500M

echo Output do job $SLURM_JOB_ID
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
./loop_omp
```

O que acontece se colocarmos -cpu-per-tasks > 2?

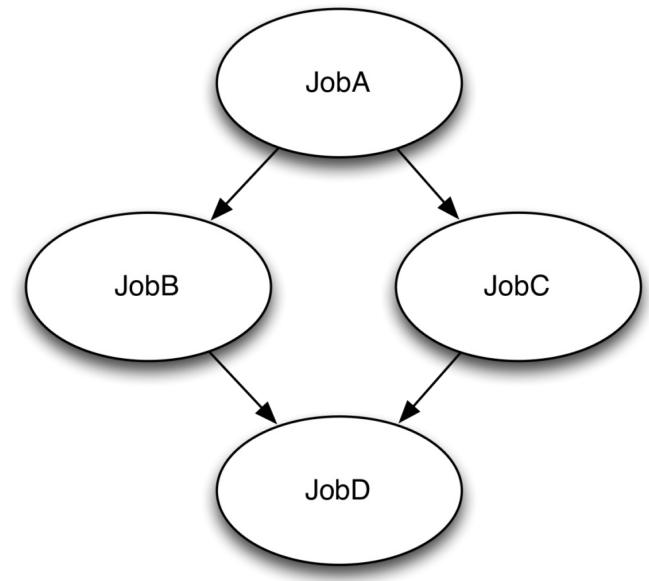
Lembre-se que no slurm o número de chips físicos a serem alocados é dado por:

nro_chips = numero de nós x numero de tasks-per-node x numero de cpus-per-task

Em um job OMP, o número de nós sempre vai ser 1. Tasks-per-node deve ser 1 também e modificamos apenas o cpus-per-tasks. Mas como fazer para rodar 4 threads sem alocar 2 nós????

Dependência entre jobs

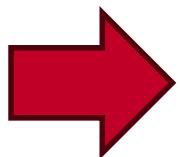
- É possível configurar dependências entre jobs e deixar com o que o SLURM determine o melhor momento de executar um job, com base no grafo de dependências.
- Vamos simular uma dependência de um novo job ao nosso loop serial



```
#include<iostream>
#include<cmath>
#include<vector>
#include<fstream>
using namespace std;

int main(){
    const int size = 50000000;
    vector<double> results(size);
    double sum = 0.0;

    for(int i = 0; i < size; ++i) {
        results[i] = sqrt(static_cast<double>(i));
        sum += results[i];
    }
    ofstream outfile("simulation_results.txt");
    for(auto &value: results){
        outfile << value << "\n";
    }
    outfile.close();
    cout << "Simulacao completa. Soma das raizes = " << sum << endl;
    return 0;
}
```



```
#include<iostream>
#include<fstream>
#include<vector>
#include<numeric>
#include<algorithm>
using namespace std;

int main(){
    vector<double> results;
    double value;
    ifstream infile("simulation_results.txt");
    while(infile >> value) {
        results.push_back(value);
    }
    infile.close();
    double sum = accumulate(results.begin(), results.end(), 0.0);
    double mean = sum / results.size();

    cout << "Análise completa. Média das raízes: " << mean << endl;
    return 0;
}
```

```
#!/bin/bash
#SBATCH --job-name=loop_serial
#SBATCH --nodes=1
#SBATCH --partition=express
#SBATCH --mem=500M

echo Output do job $SLURM_JOB_ID
./loop_serial

~
```

```
#!/bin/bash

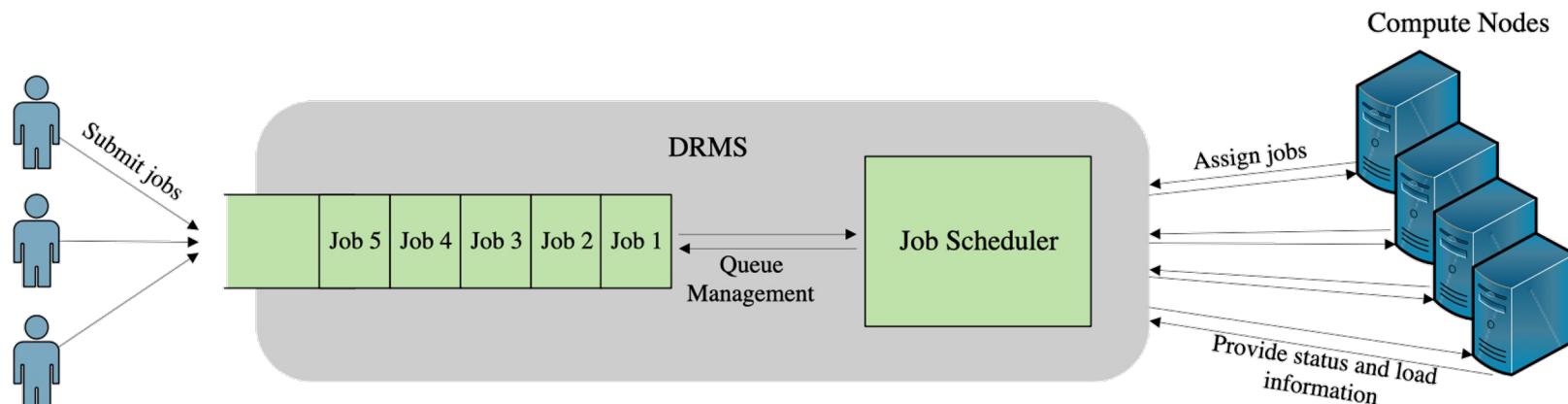
#SBATCH --job-name=data_analysis
#SBATCH --partition=express
#SBATCH --output=data_analysis.out
./data_analysis
```

```
[test@sms-host ~]$ sbatch loop_serial.slurm
Submitted batch job 48
[test@sms-host ~]$ sbatch --dependency=afterok:48 data_analysis.slurm
Submitted batch job 49
[test@sms-host ~]$ squeue
             JOBID PARTITION      NAME     USER ST       TIME  NODES NODELIST(REASON)
                  49   express data_ana   test  PD      0:00      1 (Dependency)
                  48   express loop_ser   test  R       0:19      1 compute00
[test@sms-host ~]$
```

Modificando nosso cluster – Parte II

Estamos observando que, por padrão, o SLURM aloca cada nó exclusivo a um JOB. Isto é, se temos um job que está solicitando apenas 1 core, e nosso nó possui 2 cores, ficamos com um 1 core ocioso.

Isso se dá em função do mecanismo de **scheduling** que o SLURM adota.



Modificando nosso cluster

- O que queremos mudar?
 - 1 – aumentar o número de cores de cada compute_node (faça isso no Virtualbox). Pare os nós, aumente para 3 cores por compute_node e inicialize as máquinas
 - 2 – queremos permitir que memória e cores sejam **recursos consumíveis**. Isto é, deve ser possível executar múltiplos Jobs em um nó, desde que todos os Jobs não excedam os recursos disponíveis. Para isso, vamos precisar mudar o arquivo slurm.conf, refletir esse arquivo nos nós de execução e reiniciar os deamons.

Slurm.conf (na máquina master)

```
root@sms-host:/home/vagrant
InactiveLimit=0
MinJobAge=300
KillWait=30
Waittime=0
#
# SCHEDULING
SchedulerType=sched/backfill
#SchedulerAuth=
SelectType=select/cons_res
SelectTypeParameters=CR_Core_Memory
#FastSchedule=1
#PriorityType=priority/multifactor
#PriorityDecayHalfLife=14-0
```

Aqui nós definimos que o mecanismo de alocação vai ser o de CONS_RES (recursos consumíveis) e que estes recursos são os cores da máquina e a sua memória (CR_Core_Memory)

```
Epilog=/etc/slurm/slurm.epilog.clean
NodeName=compute0[0-1] Sockets=1 CoresPerSocket=3 RealMemory=3000 ThreadsPerCore=1 State=UNKNOWN
PartitionName=normal Nodes=compute0[0-1] Default=YES MaxMemPerNode=3000 MaxTime=24:00:00 State=UP Shared=YES
PartitionName=express Nodes=compute0[0-1] Default=NO MaxMemPerNode=1000 MaxTime=00:05:00 State=UP MaxNodes=1
ReturnToService=1
```

Na linha em que definimos os nodes, garanta que CoresPerSocket seja 3.

```
Epilog=/etc/slurm/slurm.epilog.clean
NodeName=compute0[0-1] Sockets=1 CoresPerSocket=3 RealMemory=3000 ThreadsPerCore=1 State=UNKNOWN
PartitionName=normal Nodes=compute0[0-1] Default=YES MaxMemPerNode=3000 MaxTime=24:00:00 State=UP Shared=YES
PartitionName=express Nodes=compute0[0-1] Default=NO MaxMemPerNode=1000 MaxTime=00:05:00 State=UP MaxNodes=1 Shared=YES
ReturnToService=1
```

Para cada fila, garanta que temos a opção Shared=YES

Refletir as mudanças

- Desligue todos os compute nodes
- Vamos precisar copiar o slurm.conf para os nós, para que todos estejam sob a mesma configuração. Como nossos nós fazem boot por rede, você deve copiar o arquivo que está em `/etc/slurm/slurm.conf` para `/install/netboot/centos7.7/x86_64/compute/rootimg/etc/slurm/`
- Suba os nós.

Reiniciar o slurmctld

- Quando mudamos o mecanismo de scheduling do SLURM ele entra em um estado inconsistente em relação aos Jobs que já estavam executando. Dessa forma, há uma grande chance do slurmctld não iniciar, apresentando erro.
- Esse erro é conhecido e há uma KB sobre ele, disponível [aqui](#)
- Mas na prática para resolver isso temos que fazer:
 - Iniciar o slurm com a opção -i: `/usr/sbin/slurmctld -i`
 - Executar o comando: `killall slurmctld`
 - Iniciar o slurm normalmente: `systemctl start slurmctld`

Vamos testar?

- Você pode submeter o job `loop_serial.slurm` diversas vezes. Modifique o script para ele fique da seguinte forma:
- Submeta uns 8 Jobs, e você deve ver algo parecido:

Se nosso cluster possui 6 cores, 6 Jobs devem executar simultaneamente!

```
#!/bin/bash
#SBATCH --job-name=loop_serial
#SBATCH -c 1
#SBATCH --mem=500M

echo Output do job $SLURM_JOB_ID
./loop_serial
```

```
[test@sms-host ~]$ squeue
   JOBID PARTITION     NAME     USER ST      TIME  NODES NODELIST(REASON)
       98    normal loop_ser test  PD      0:00      1 (Resources)
       99    normal loop_ser test  PD      0:00      1 (Priority)
       92    normal loop_ser test   R      0:05
       93    normal loop_ser test   R      0:04
       94    normal loop_ser test   R      0:04
       95    normal loop_ser test   R      0:02
       96    normal loop_ser test   R      0:02
       97    normal loop_ser test   R      0:02
```

Vamos fazer o cluster 'trabalhar'!

Crie o arquivo ao lado, chamado long_fibo.cpp

Compile com g++
(não se esqueça que há flags omp nele!)

```
#include<iostream>
#include<vector>
#include<omp.h>

long long fibonacci(int n){
    if (n <=1) {
        return n;
    } else {
        return fibonacci(n-1) + fibonacci(n-2);
    }
}

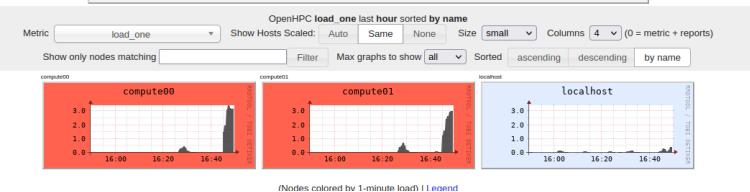
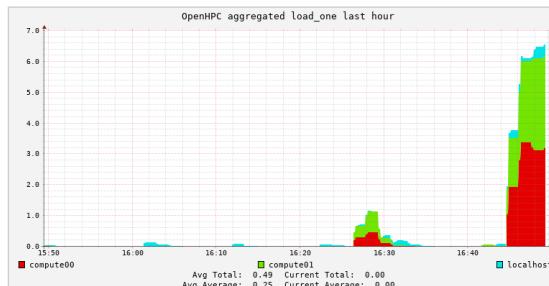
int main() {
    const int MAX_FIB = 100;
    #pragma omp parallel
    {
        #pragma omp for
        for(int i = 0; i < MAX_FIB; ++i){
            long long fib_value = fibonacci(i);
            #pragma omp critical
            {
                std::cout<< " Fibonacci( " << i << " ) = " << fib_value << std::endl;
            }
        }
    }
    return 0;
}
```

Vamos fazer o cluster 'trabalhar'!

Faça um arquivo de submissão como segue:

```
#!/bin/bash
#SBATCH --job-name=long_fibo
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=200M

echo Output do job $SLURM_JOB_ID
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
./long_fibo
```

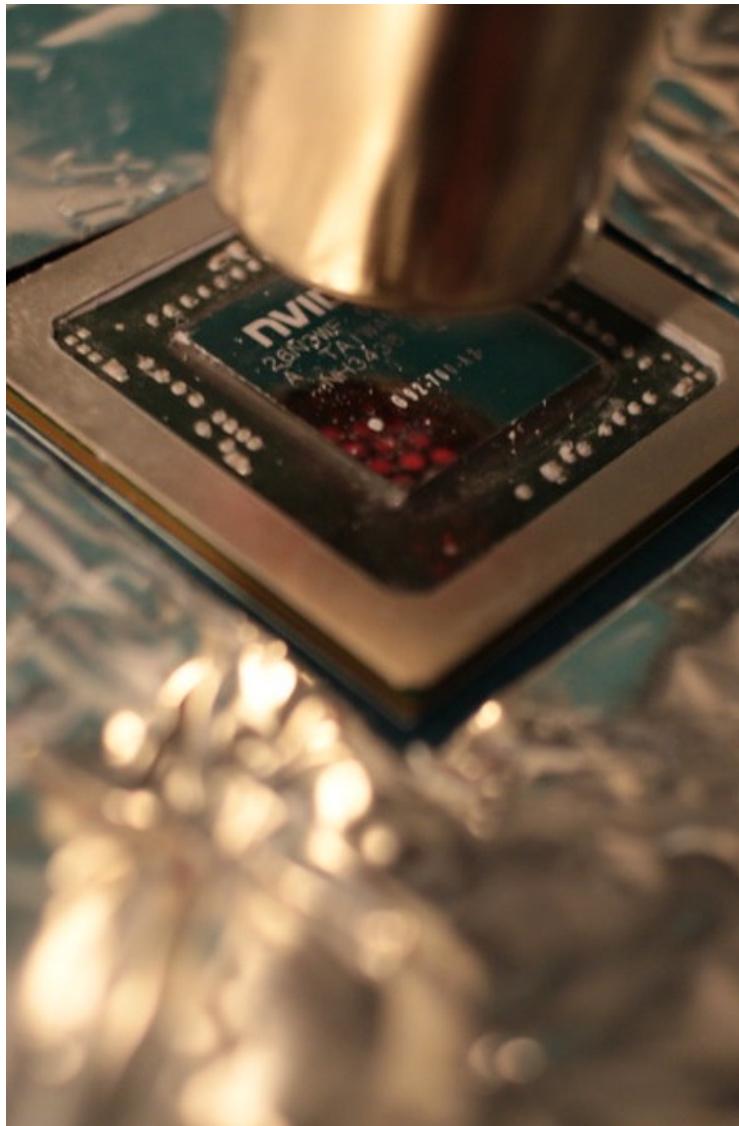


Exercício

- Modifique o arquivo de submissão para aumentar o número de threads. Submeta Jobs de modo que se possível testar o desempenho serial, com 2 threads e com 3 threads.
- Busque controlar a fila o mais adequado possível de modo que ao menos 2 Jobs executem simultaneamente
- Coloque wall-time de 30 minutos nos Jobs. Observe nos arquivos de output até qual valor de n o Fibonacci foi calculado
- Monitore no Ganglia



Insper Supercomputação



MPI

- Introdução ao MPI (*Message Passing Interface*)

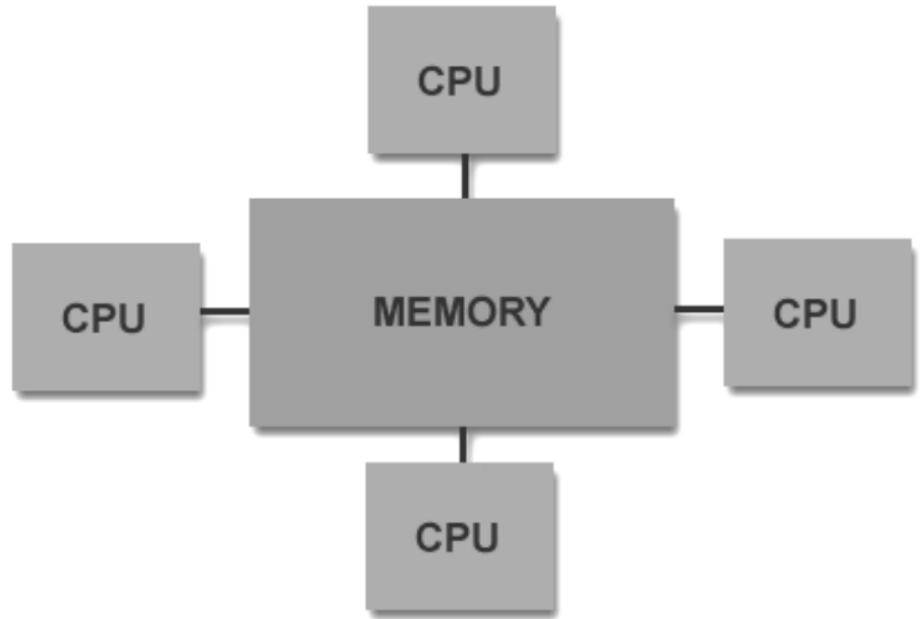
O que vimos até hoje?

- Temos um “problemão” para resolver (muitos dados e/ou muitos cálculos)
- Como podemos tratar o problema?

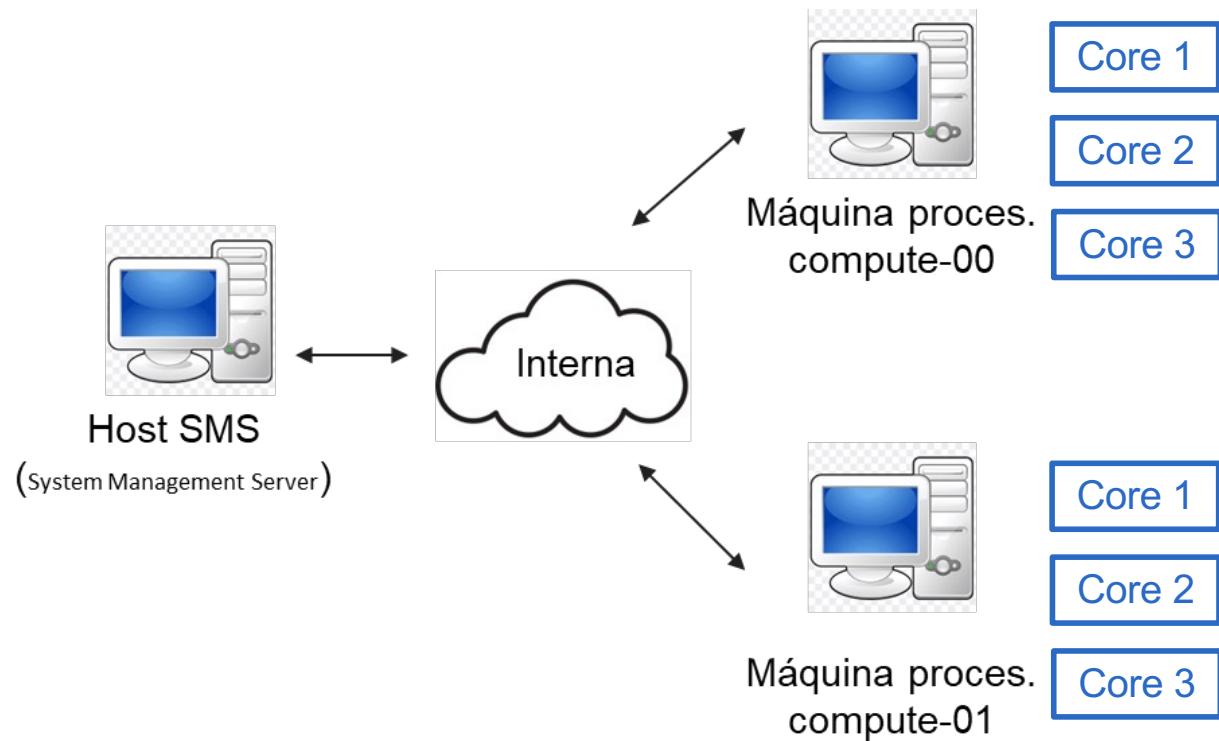


Até onde vai OpenMP?

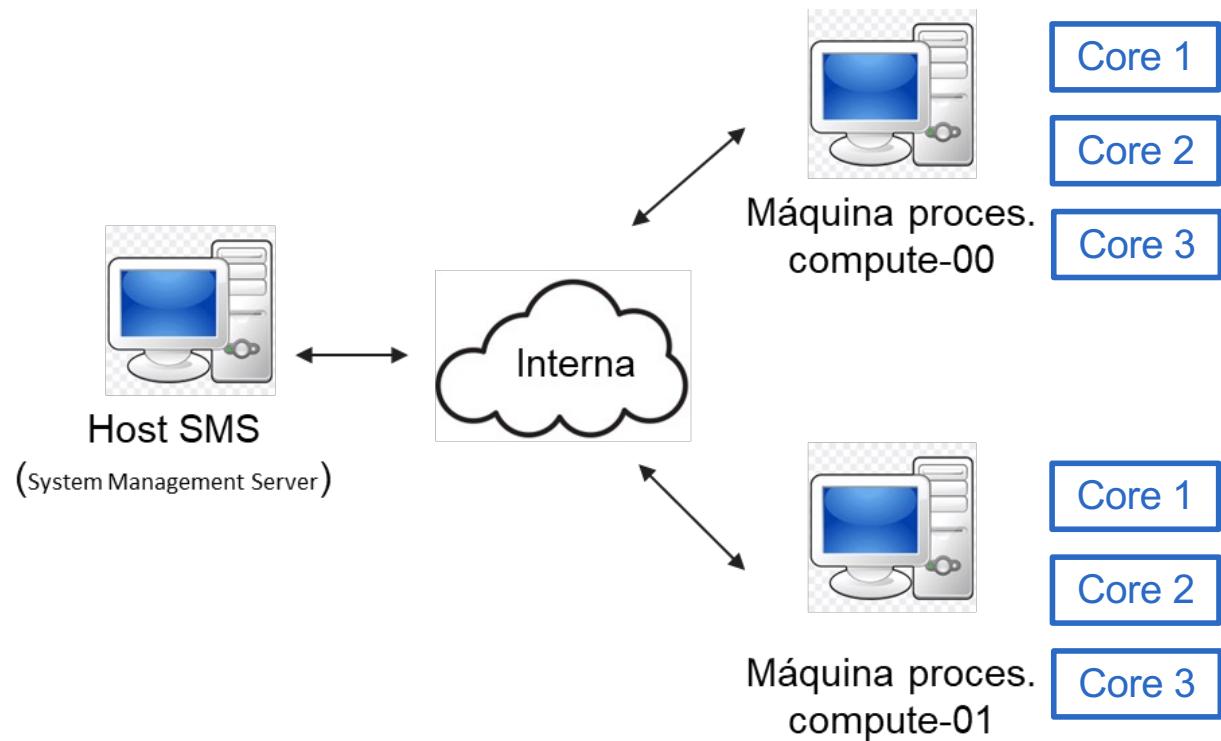
- Focado em ambientes de memória compartilhada
- “Paralelismo local”
- Estou “usando ao máximo” a **máquina** que tenho



E o nosso cluster?



E o nosso cluster?

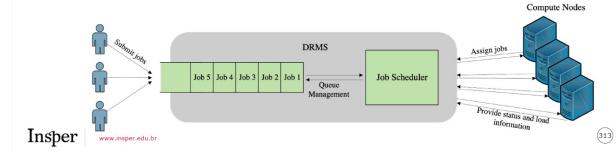


Funcionamento padrão

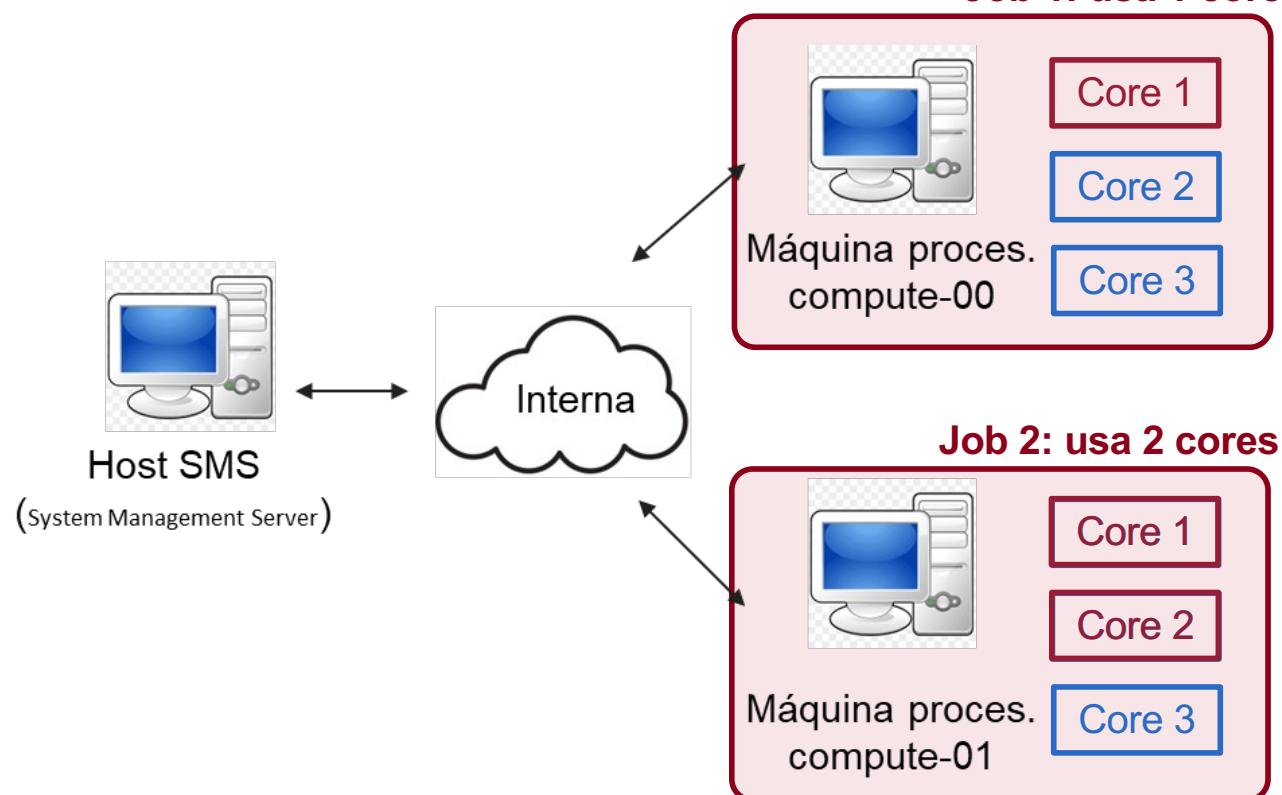
Modificando nosso cluster – Parte II

Estamos observando que, por padrão, o SLURM aloca cada nó exclusivo a um JOB. Isto é, se temos um job que está solicitando apenas 1 core, e nosso nó possui 2 cores, ficamos com um 1 core ocioso.

Isso se dá em função do mecanismo de **scheduling** que o SLURM adota.



E o nosso cluster?

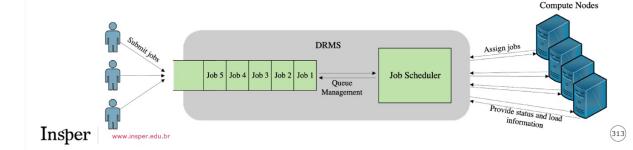


Funcionamento padrão

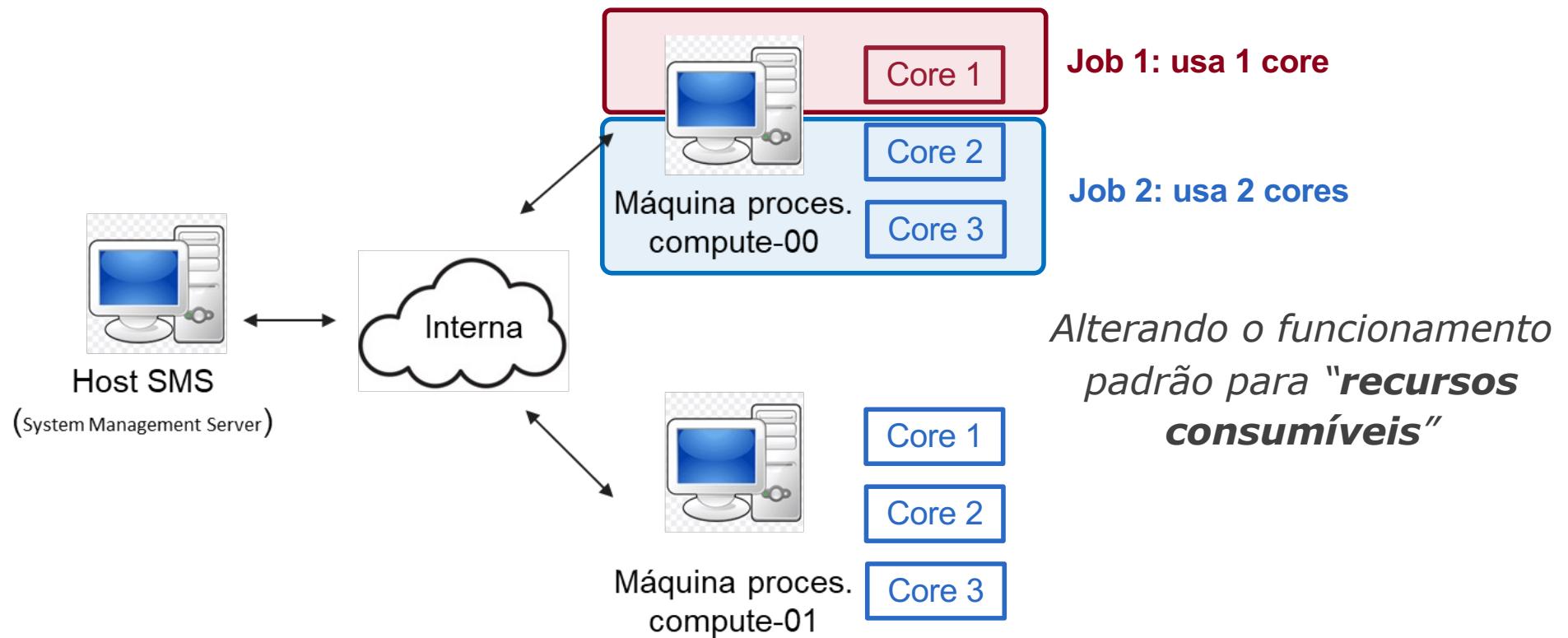
Modificando nosso cluster – Parte II

Estamos observando que, por padrão, o SLURM aloca cada nó exclusivo a um JOB. Isto é, se temos um job que está solicitando apenas 1 core, e nosso nó possui 2 cores, ficamos com um 1 core ocioso.

Isso se dá em função do mecanismo de **scheduling** que o SLURM adota.



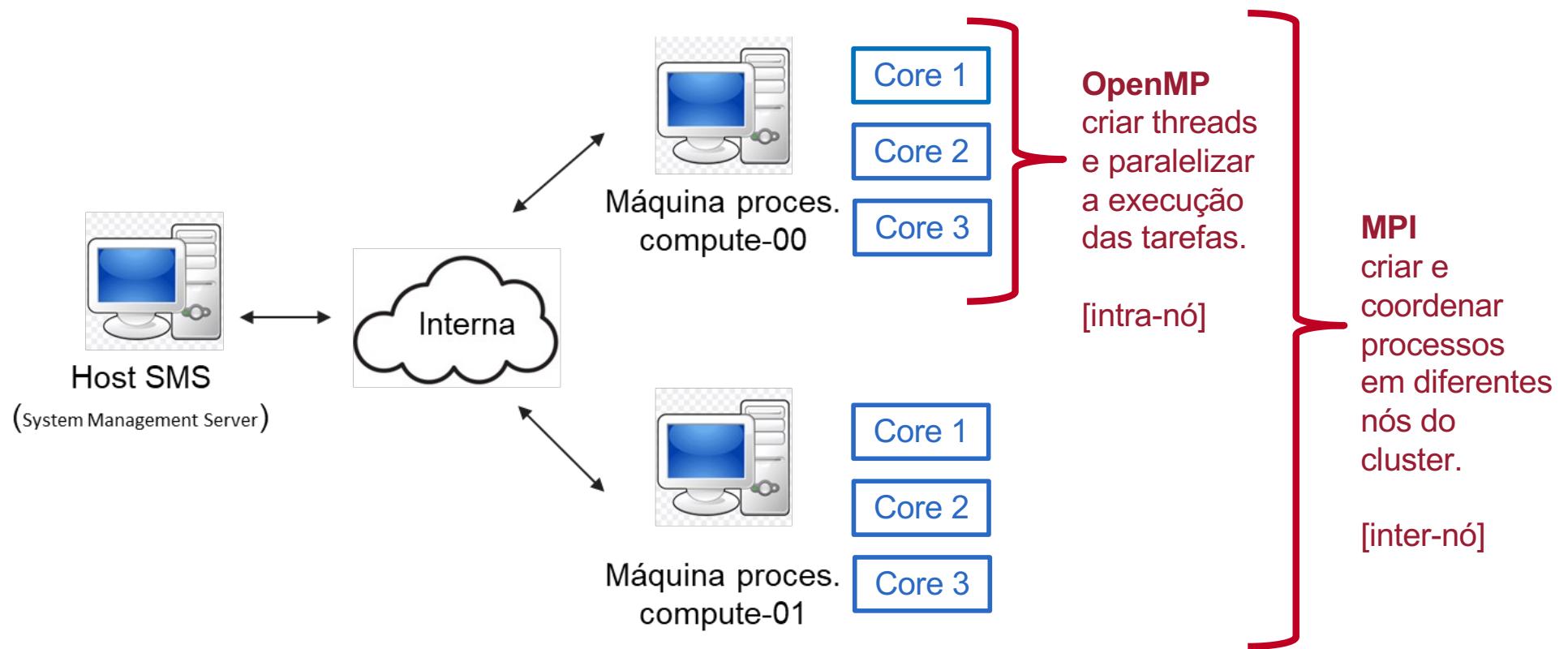
E o nosso cluster?



Então...

- **OpenMP**: paralelismo local, com memória compartilhada
- **Cluster**: paralelismo de tarefas, em memória distribuída
- Podemos combinar ambos? SIM!
 - MPI

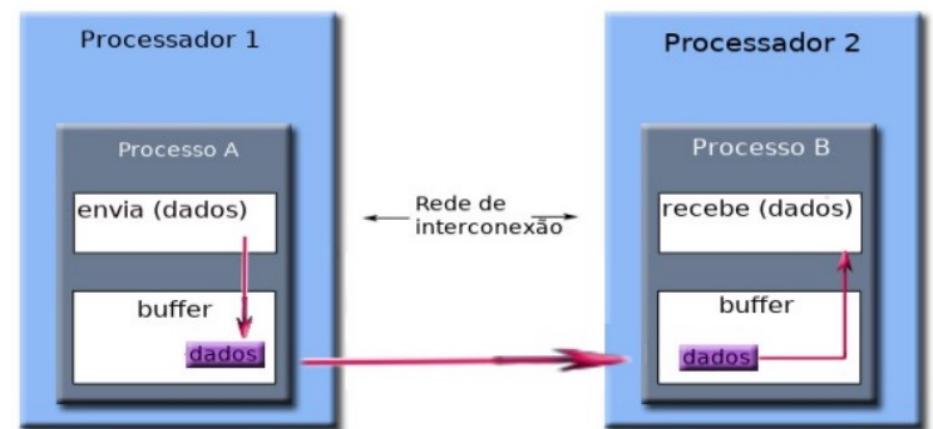
Combinando OpenMP com MPI



MPI : Message Passing Interface

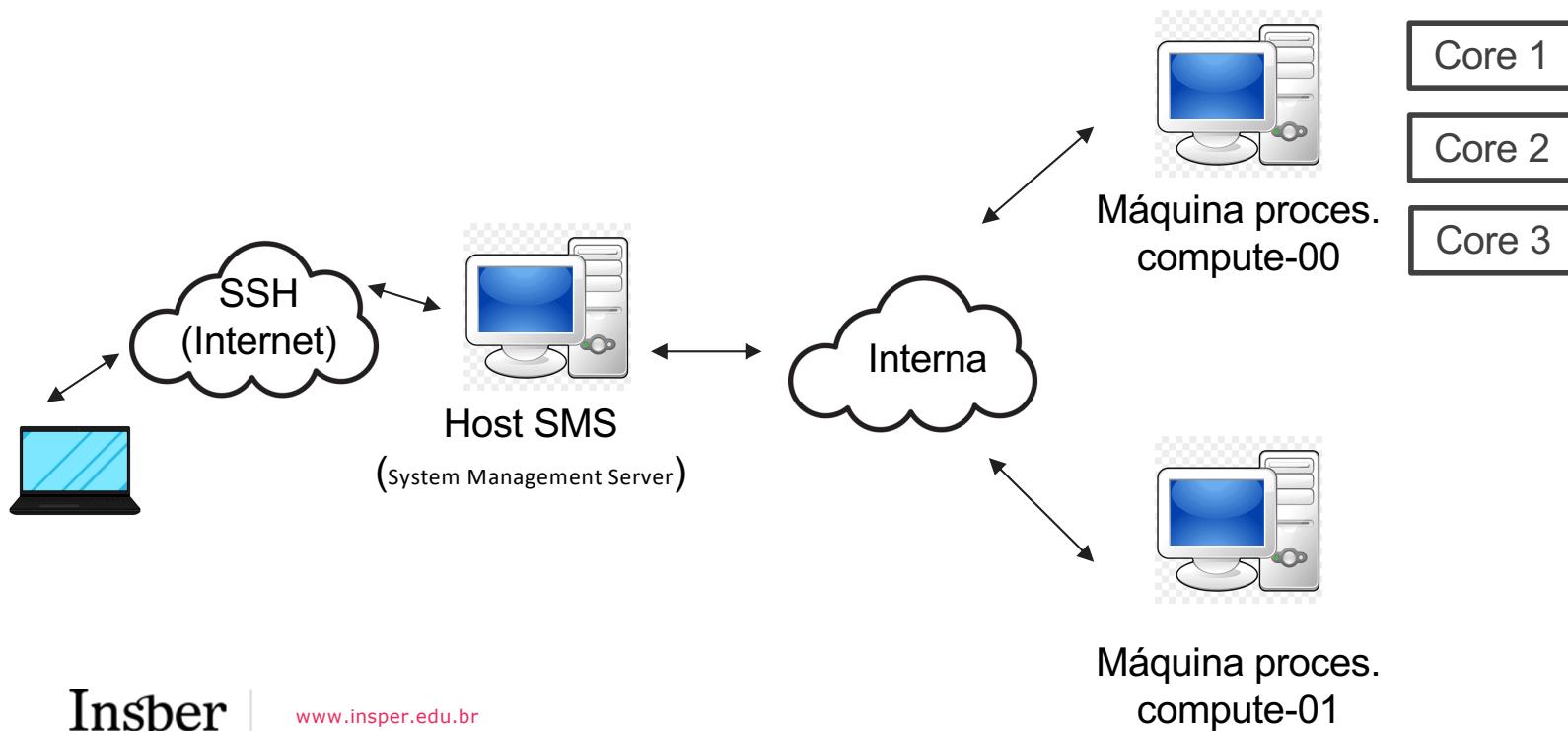
- É um padrão da indústria, não uma linguagem/implementação
- Assume que não há compartilhamento de memória
- O programador deve:
 - Dividir os dados
 - Trabalhar com interações são bilaterais (*send / receive*)
 - Reduzir comunicação (quando possível), pra otimizar desempenho

Posso usar MPI localmente?
(em ambiente de memória compartilhada)



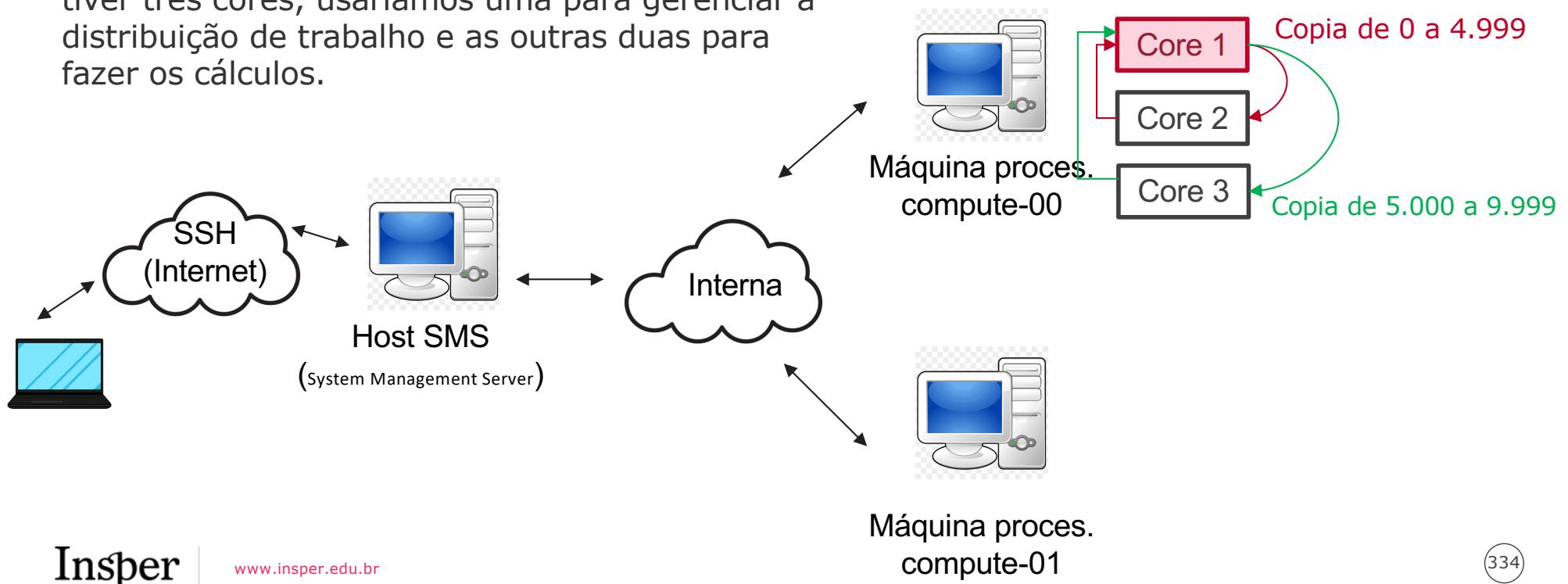
MPI : Exemplo (local)

```
1 | int s = 0, v[ ] = {1,2,3,4};  
2 |  
3 | for (int i=0; i<4; i++)  
4 |     s += v[i];
```



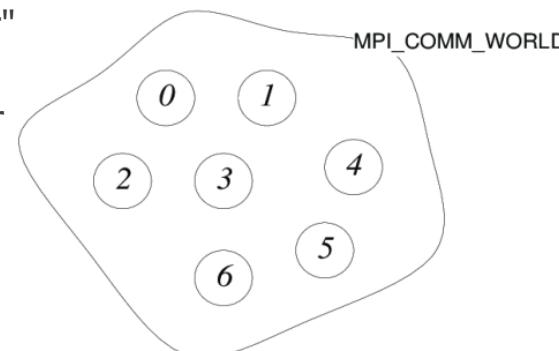
MPI : Exemplo (local)

Se o vetor tivesse 10.000 entradas e a máquina tiver três cores, usariamos uma para gerenciar a distribuição de trabalho e as outras duas para fazer os cálculos.

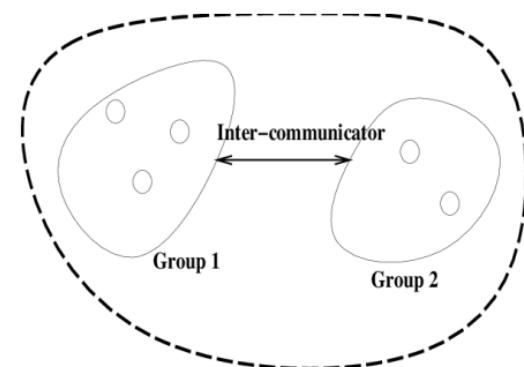


MPI : Conceitos

- **Rank:** Todo processo tem uma única identificação, atribuída pelo sistema quando o processo é iniciado. Essa identificação é continua e começa no 0 até n-1 processos.
- **Group:** Grupo é um conjunto ordenado de N processos. Todo e qualquer grupo é associado a um "communicator" e, inicialmente, todos os processos são membros de um grupo com um "communicator" já pré-estabelecido (**MPI_COMM_WORLD**).
- **Communicator:** O "communicator" define uma coleção de processos (grupo), que poderão se comunicar entre si (contexto). O MPI utiliza essa combinação de grupo e contexto para garantir uma comunicação segura e evitar problemas no envio de mensagens entre os processos.



MacDonald et al. (2020) Fagg et al. (1997)



MPI : Código | Execução

Include

```
#include<mpi.h>      mpic++ programa.cpp -o programa
```

Compilação

Execução (básica)

```
mpirun -np <num processos> ./programa <argumentos do programa>
```

Na execução:

- **-np** especifica quantidade de processos a serem criados; cada um executa uma cópia do executável (SPMD)
- Há limites para **-np**: quantidade de slots disponíveis na arquitetura onde o executável será executado. **Slots** representam quantidade de processadores físicos disponíveis.
- **--use-hwthread-cpus** permite usar também os processadores lógicos
- **--oversubscribe** permite mais de um processo por **slot**
- **--hostfile <hostfilename>**, especifica nós que podem ser usados para atribuir os processos gerados (mapeamento de processos em processadores).
 - O parâmetro **hostfilename** é um arquivo texto com endereços ou nomes dos nós (hosts/máquinas). Um hostname por linha.
 - Os nós podem ser especificados sem o **--hostfile** (usar **-host-host-H**)

MPI : Tipos de dados

Tipo do MPI	Tipo do C
MPI_CHAR	char
MPI_SHORT	short int
MPI_INT	int
MPI_LONG	long int
MPI_LONG_LONG_INT	long long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_UNSIGNED_LONG_LONG	unsigned long long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_WCHAR	wide char
MPI_PACKED	special data type for packing
MPI_BYTE	single byte value

MPI : Funções básicas

- int MPI_Init(int *argc, char ***argv)
- int MPI_Finalize()
- int MPI_Comm_size(MPI_Comm comm, int *size)
- int MPI_Comm_rank(MPI_Comm comm, int *rank)
- int MPI_Send(void bufferE, int count, MPI_Datatype datatype, int dest, int tag,
 MPI_Comm comm)
- int MPI_Recv(void bufferR, int count, MPI_Datatype datatype, int source, int tag,
 MPI_Comm comm, MPI_Status status).

MPI : Exemplo

Mensagem=dado(3 parâmetros de informações) + envelope(3 parâmetros de informações)

Ex.: CALL MPI_SEND(sndbuf, count, datatype, dest, tag, comm, mpierr)
 DADO ENVELOPE

CALL MPI_RECV(recvbuf, count, datatype, source, tag, comm, status, mpierr)
 DADO ENVELOPE

- mpcc hello.c -o hello
- mpirun -np 2 ./hello

C

```
#include <stddef.h>
#include <stdlib.h>
#include "mpi.h"
main(int argc, char **argv )
{
    char message[20];
    int i,rank, size, type=99;
    MPI_Status status;

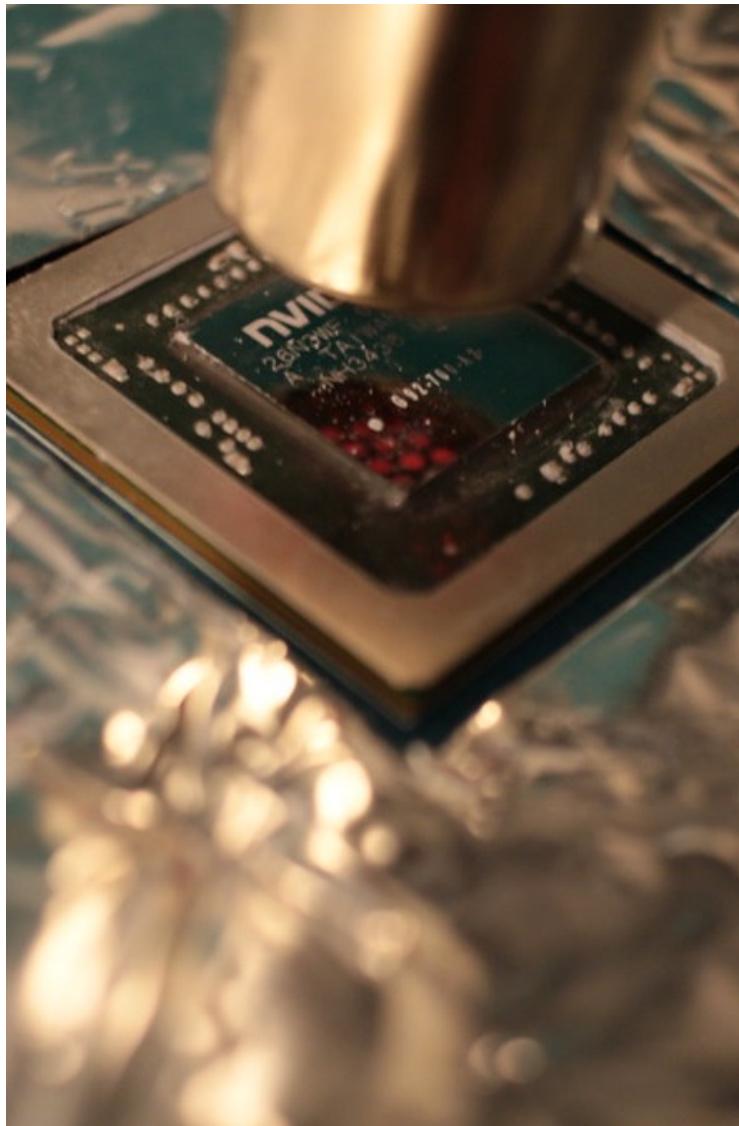
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD,&size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    if(rank == 0) {
        strcpy(message, "Hello, world");
        for (i=1; i<size; i++)
            MPI_Send(message, 13, MPI_CHAR, i,
                     type, MPI_COMM_WORLD);
    }
    else
        MPI_Recv(message, 13, MPI_CHAR, 0,
                 type, MPI_COMM_WORLD, &status);

    printf( "Message from node =%d : %.13s\n",
            rank,message);
    MPI_Finalize();
}
```



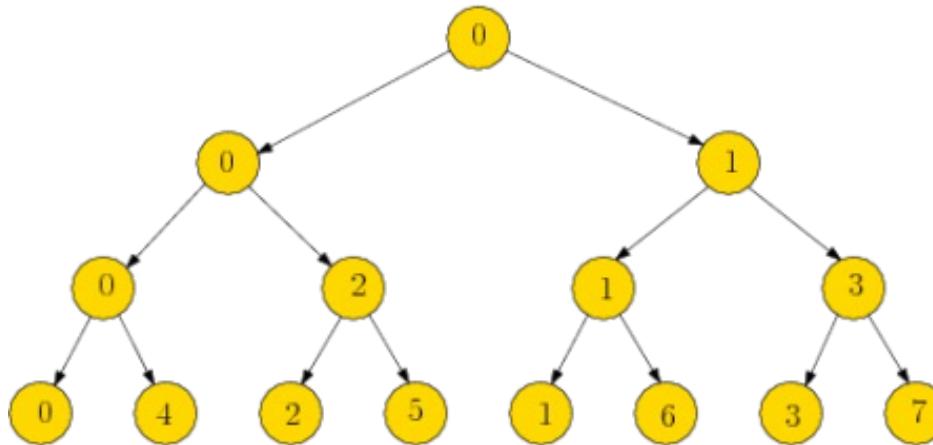
Insper Supercomputação



MPI – Comunicação

- Modos de Comunicação Ponto-a-Ponto

Broadcast e Reduce



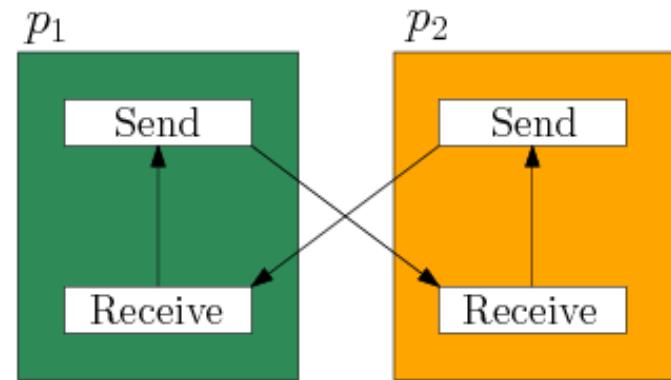
- Mandar uma mensagem para todos os processos: *MPI_Bcast*
- Receber mensagens dos nós filhos: *MPI_Reduce*

Formas de comunicação

- Bloqueante vs não bloqueante
- Aspectos de hardware influenciam fortemente:
 - Há buffers associados ao send ou ao receive?
 - Há um hardware específico para realizar a comunicação em paralelo à CPU?

Formas de comunicação

- A forma mais comum de comunicação entre dois processos no MPI é através de Sends e Receives, implementados pelas funções `MPI_Send` e `MPI_Recv`, respectivamente.
- Em teoria, são bloqueantes



Formas de comunicação

- A maior parte das implementações do padrão MPI, provê um **buffer** para que o send não seja bloqueante.
- Ao chamar *MPI_Send*, a mensagem é copiada em um buffer e a execução do programa continua. Aí quando o destinatário chama *MPI_Recv*, ele lê a mensagem do buffer.
- Em implementações onde não há o esse buffer, deve-se usar *MPI_Bsend*, onde o buffer deve ser explicitamente alocado.

Modos de comunicação ponto a ponto

- O MPI possui 08 *sends* e 02 *receives* (semânticas distintas)
- Há 04 modos para o *send*:
 - **Standard**
 - Pode ou não ter buffer interno do MPI. Transferência síncrona ou assíncrona.
 - Implementação decide
 - **Buffered**
 - O usuário cria previamente um buffer (espaço usuário) e o utiliza para o *send*
 - **Synchronous**
 - Exige a sincronização entre o *send* e o início da execução do *receive* no mínimo
 - Depende se usa buffer e se é bloqueante ou não bloqueante
 - **Ready**
 - *Receive* deve ser executado antes do *send*. Programador garante isso
- Cada um destes modos pode ser bloqueante ou não bloqueante (temos 08 *sends*)
- Há 01 modo para o *receive*:
 - **Standard**
 - Análogo ao *send standard*, porém, para o recebimento da mensagem
 - O *receive standard* pode ser bloqueante ou não bloqueante (temos 02 *receives*)

Modos de comunicação ponto a ponto

- Sends bloqueantes no MPI

- **Standard**

```
int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
```

- **Buffered**

```
int MPI_Bsend(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
```

- **Synchronous**

```
int MPI_Ssend(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
```

- **Ready**

```
int MPI_Rsend(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
```

Modos de comunicação ponto a ponto

- *Buffered (um pouco mais de detalhe)*

```
int MPI_Bsend(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
```

- O buffer precisa ser criado pelo programador
- *int MPI_Pack_size(int incount, MPI_Datatype datatype, MPI_Comm comm, int *size)*
 - Retorna em ***size** o limite superior necessário para empacotar msg no buffer
 - Buffer pode ser usado por mais de um **MPI_Bsend**. Neste caso considera o total
- *int MPI_Buffer_attach(void *buf, int size)*
 - Associa um buffer ao MPI no espaço do usuário para enviar msgs
 - Deve considerar: **MPI_BSEND_OVERHEAD** por mensagem que usar o buffer
- *MPI_Bsend()*
 - Envia o conteúdo do buffer indicado em **MPI_Bsend()**
- *int MPI_Buffer_detach(void *buf, int *size)*
 - Desassocia o buffer do MPI e espera o término de msgs que estejam usando o buffer
 - Não desaloca da memória, apenas desassocia do MPI

Modos de comunicação ponto a ponto

- Sends não bloqueantes no MPI

- **Standard**

```
int MPI_Isend(void *buf, int count, MPI_Datatype dtype, int dest, int tag, MPI_Comm comm,  
MPI_Request *request);
```

- **Buffered**

```
int MPI_Ibsend(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm  
comm, MPI_Request *request)
```

- **Synchronous**

```
int MPI_Issend(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm  
comm, MPI_Request *request)
```

- **Ready**

```
int MPI_Irsend(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm  
comm, MPI_Request *request)
```

- O parâmetro de saída ***request é um handle** que identifica a msg não bloq em andamento
 - Permite verificar se a comunicação já foi finalizada com o **MPI_Test** ou **MPI_Wait**

Modos de comunicação ponto a ponto

- Receives bloqueantes e não bloqueantes no MPI (apenas **Standard**)

- **Receive Bloqueante**

```
int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)
```

- **Receive Não Bloqueante**

```
int MPI_Irecv(void *buf, int count, MPI_Datatype dtype, int source, int tag, MPI_Comm comm, MPI_Request *request)
```

- **MPI_Get_count(MPI_Status *status, MPI_Datatype datatype, int *count)**

- Pode ser usada com ***status** para obter informações da msg recebida

Modos de comunicação ponto a ponto

- Há primitivas testam e/ou esperam o fim de primitivas não bloqueantes no MPI
- *int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)*
 - Faz um teste não bloqueante da primitiva não bloqueante indicada por ***request**
 - O parâmetro ***flag** tem esse retorno
- *int MPI_Wait(MPI_Request *request, MPI_Status *status)*
 - Faz um teste bloqueante da primitiva não bloqueante indicada por ***request**
 - Se a mensagem ainda não estiver segura, o processo chamador de **MPI_Wait** bloqueia até que a mensagem esteja segura.
- *MPI_Get_count(MPI_Status *status, MPI_Datatype datatype, int *count)*
 - Pode ser usada com ***status** para obter informações da msg recebida

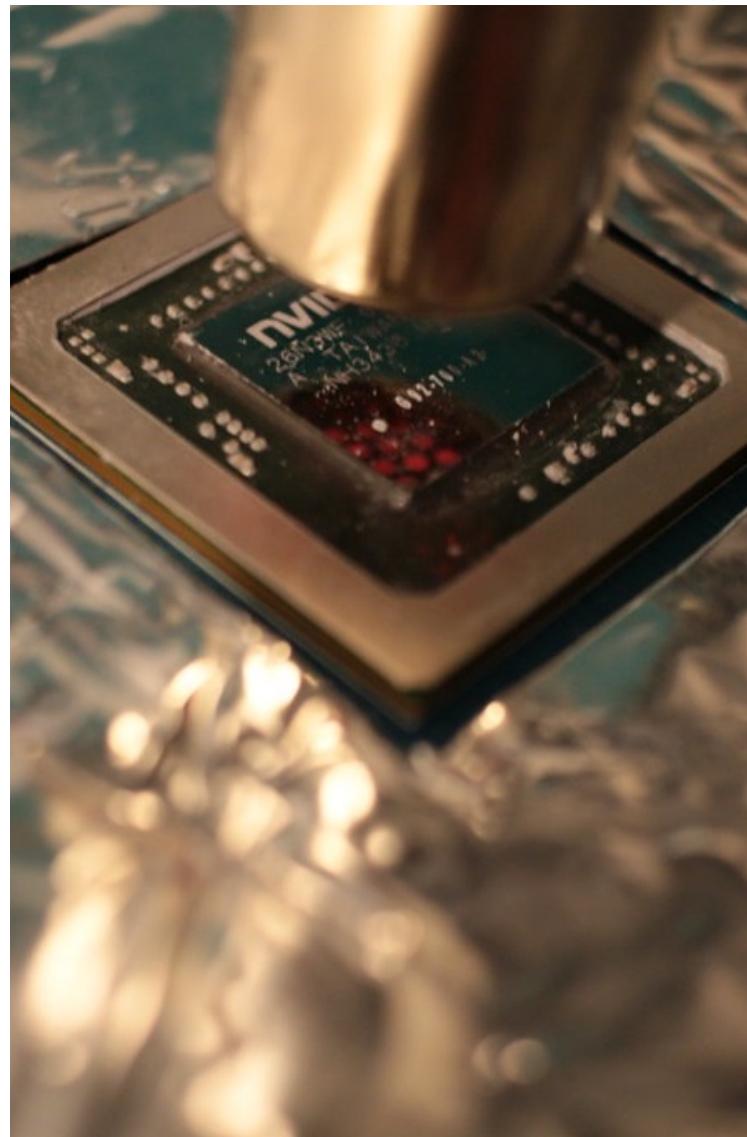
Prática

<https://encurtador.com.br/vyDO6>

1. Submeter o “Hello, World!” de MPI no cluster com slurm
2. Submeter um job de MPI que sobrecarrega a comunicação no cluster, observando o uso da rede no Ganglia
3. Faça você mesmo:
 1. Crie um programa via C++/MPI que calcula a soma de um vetor de 10k números.
 2. Submeta este programa no cluster via slurm:
 1. Faça uma execução local (usando apenas um node)
 2. Faça uma execução global (usando todos os nodes e recursos disponíveis).
 3. Observe e compare os tempos de execução



Insper Supercomputação



MPI

- Scatter e Gather

Revisitando

- Broadcast

`MPI_Bcast(buffer, count, datatype, root, comm)`

buffer

data to be distributed

root

rank of broadcast root

count

number of entries in buffer

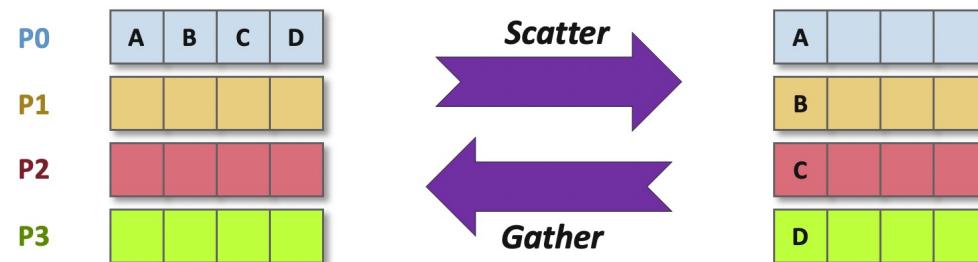
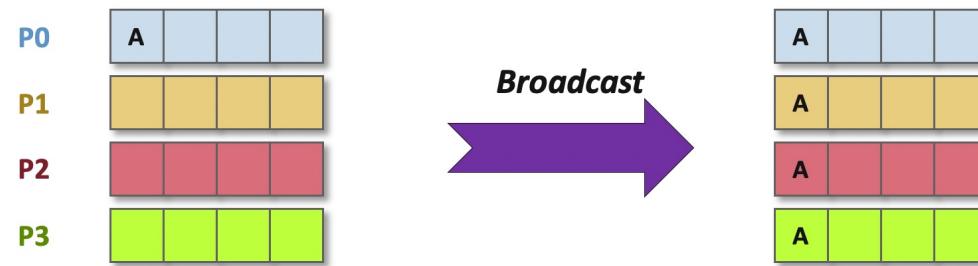
comm

communicator

datatype

data type of buffer

Broadcast vs Scatter / Gather



Scatter

- Task root sends an equal share of data to all other processes

`MPI_Scatter(sendbuf, sendcount, sendtype, recvbuf, recvcount,
recvtype, root, comm)`

sendbuf

send buffer (data to be scattered)

sendcount

number of elements sent to each process

sendtype

data type of send buffer elements

recvbuf

receive buffer

recvcount

number of elements to receive at each process

recvtype

data type of receive buffer elements

root

rank of sending process

comm

communicator

Exemplo de Código SEND / RECV

- Enviando um vetor

```
#include <iostream>
#include <mpi.h>

int main(int argc, char *argv[]) {
    MPI_Init(&argc, &argv);

    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    const int ARRAY_SIZE = 100;
    int array[ARRAY_SIZE];

    if (rank == 0) {
        for (int i = 0; i < ARRAY_SIZE; i++) {
            array[i] = i;
        }
    }

    int chunk_size = ARRAY_SIZE / size;
    int recv_array[chunk_size];

    if (rank == 0) {
        for (int i = 1; i < size; i++) {
            MPI_Send(&array[i * chunk_size], chunk_size, MPI_INT, i, 0, MPI_COMM_WORLD);
        }
    } else {
        MPI_Recv(recv_array, chunk_size, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        // Imprime o vetor recebido
        std::cout << "Nº " << rank << " recebeu: ";
        for (int i = 0; i < chunk_size; i++) {
            std::cout << recv_array[i] << " ";
        }
        std::cout << std::endl;
    }

    MPI_Finalize();
    return 0;
}
```

Exemplo de Código SCATTER

- Enviando um vetor

```
#include <iostream>
#include <mpi.h>

int main(int argc, char *argv[]) {
    MPI_Init(&argc, &argv);

    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    const int ARRAY_SIZE = 100;
    int array[ARRAY_SIZE];

    if (rank == 0) {
        for (int i = 0; i < ARRAY_SIZE; i++) {
            array[i] = i;
        }
    }

    int chunk_size = ARRAY_SIZE / size;
    int recv_array[chunk_size];

    MPI_Scatter(array, chunk_size, MPI_INT, recv_array, chunk_size, MPI_INT, 0, MPI_COMM_WORLD);

    // Imprime o vetor recebido
    std::cout << "Nó " << rank << " recebeu: ";
    for (int i = 0; i < chunk_size; i++) {
        std::cout << recv_array[i] << " ";
    }
    std::cout << std::endl;

    MPI_Finalize();
    return 0;
}
```

MPI e OpenMP

- **Tarefa:** Calcular o quadrado de cada elemento em um array bidimensional.
- **MPI:** Divide o array entre diferentes processos.
- **OpenMP:** Paraleliza o cálculo dentro de cada processo.

Exemplo de código

```
#include <iostream>
#include <mpi.h>
#include <omp.h>

int main(int argc, char *argv[]) {
    MPI_Init(&argc, &argv);

    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    const int N = 10; // Dimensões do array bidimensional
    int data[N][N];

    // Inicialização do array pelo processo 0
    if (rank == 0) {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                data[i][j] = i + j;
            }
        }
    }
}
```

```
// Dividir o array entre os processos
int chunk_size = N / size;
int local_data[chunk_size][N];
MPI_Scatter(data, chunk_size * N, MPI_INT, local_data, chunk_size * N, MPI_INT, 0, MPI_COMM_WORLD);

// Paralelização com OpenMP
#pragma omp parallel for collapse(2)
for (int i = 0; i < chunk_size; i++) {
    for (int j = 0; j < N; j++) {
        local_data[i][j] *= local_data[i][j]; // Calcula o quadrado do elemento
    }
}

// Reunir os resultados no processo 0
MPI_Gather(local_data, chunk_size * N, MPI_INT, data, chunk_size * N, MPI_INT, 0, MPI_COMM_WORLD);

// Processo 0 imprime os resultados
if (rank == 0) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            std::cout << data[i][j] << " ";
        }
        std::cout << std::endl;
    }
}

MPI_Finalize();
return 0;
}
```