

SuperComputação

Aula 1 – Introdução ao C++

2019 – Engenharia

Luciano Soares [<lpsoares@insper.edu.br>](mailto:lpsoares@insper.edu.br)

Igor Montagner [<igorsm1@insper.edu.br>](mailto:igorsm1@insper.edu.br)

Hoje

- Resumo geral do curso
- Burocracias
- Arquiteturas modernas de CPU
- C++

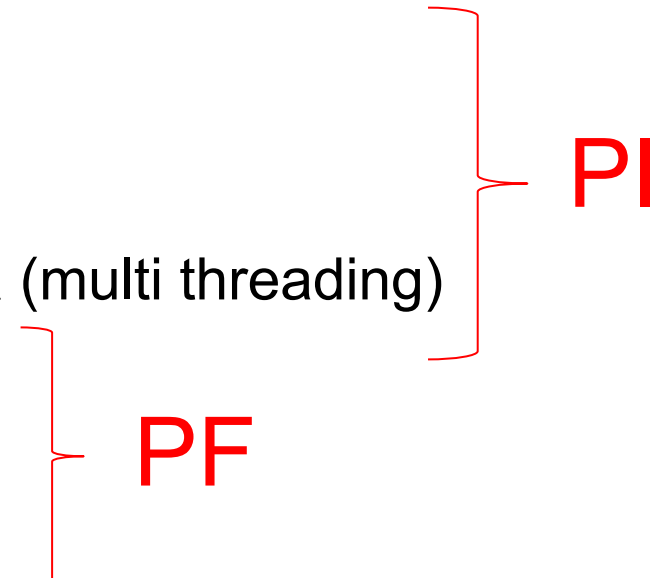
Objetivos de aprendizagem (I)

- Desenvolver algoritmos usando recursos de computação paralela/distribuída para ganhos de desempenho da aplicação final;
- Aplicar estrutura lógica de computação distribuída para o desenvolvimento de algoritmos multitarefas;
- Usar GPGPU para computação numérica e comparar com soluções baseadas em CPU

Objetivos de aprendizagem (II)

- Planejar e projetar sistemas de computação de alto desempenho;
- Analisar a complexidade dos algoritmos paralelos e a eficiência de uma implementação particular, identificando as medidas de desempenho mais adequadas para esta tarefa;
- Aplicar recursos específico de sistemas operacionais para melhorar o desempenho de algoritmos;
- Desenvolver aplicações que utilizam protocolos otimizados para paralelização.

Plano de aulas

- Revisão C++
 - SIMD
 - Memória Compartilhada (multi threading)
 - GPU
 - Computação distribuída
- PI
- PF
- 

Plano de aulas

- Revisão C++
- SIMD
- Memória Compartilhada (multi threading)
- GPU
- Computação distribuída

Projetos
incrementais

↳ Projeto menor

Burocracias

Existe uma hierarquia de entregáveis

- Tarefas apresentam conteúdo de maneira isolada
- Projetos apresenta uma situação e pede que uma série de técnicas sejam aplicadas neste contexto.
- Prova verifica que o aluno é capaz de aplicar a técnica (correta) dado um determinado contexto.

Burocracias

- Média Final:
 - Tarefas = 10%
 - Projetos = 40%
 - Provas = 50%
- Conceito I em qualquer projeto implica em reprovação
- Só é permitido tirar D em um dos projetos

Burocracias (projetos)

- Projetos serão intensos (1 semana)
- 2 / 3 aulas estúdio + 1 atendimento
- Enunciado liberado na segunda-feira
- Entrega para o próximo domingo
- Datas anunciadas no início de cada módulo

Planejem-se para que possam se dedicar ao projeto!

Burocracias

- Horário de atendimento:

Sexta-feira 9:30-11:00

- Github da disciplina

<http://github.com/insper/supercomp>

- Entregas (e notas) e avisos no Blackboard

Ferramentas

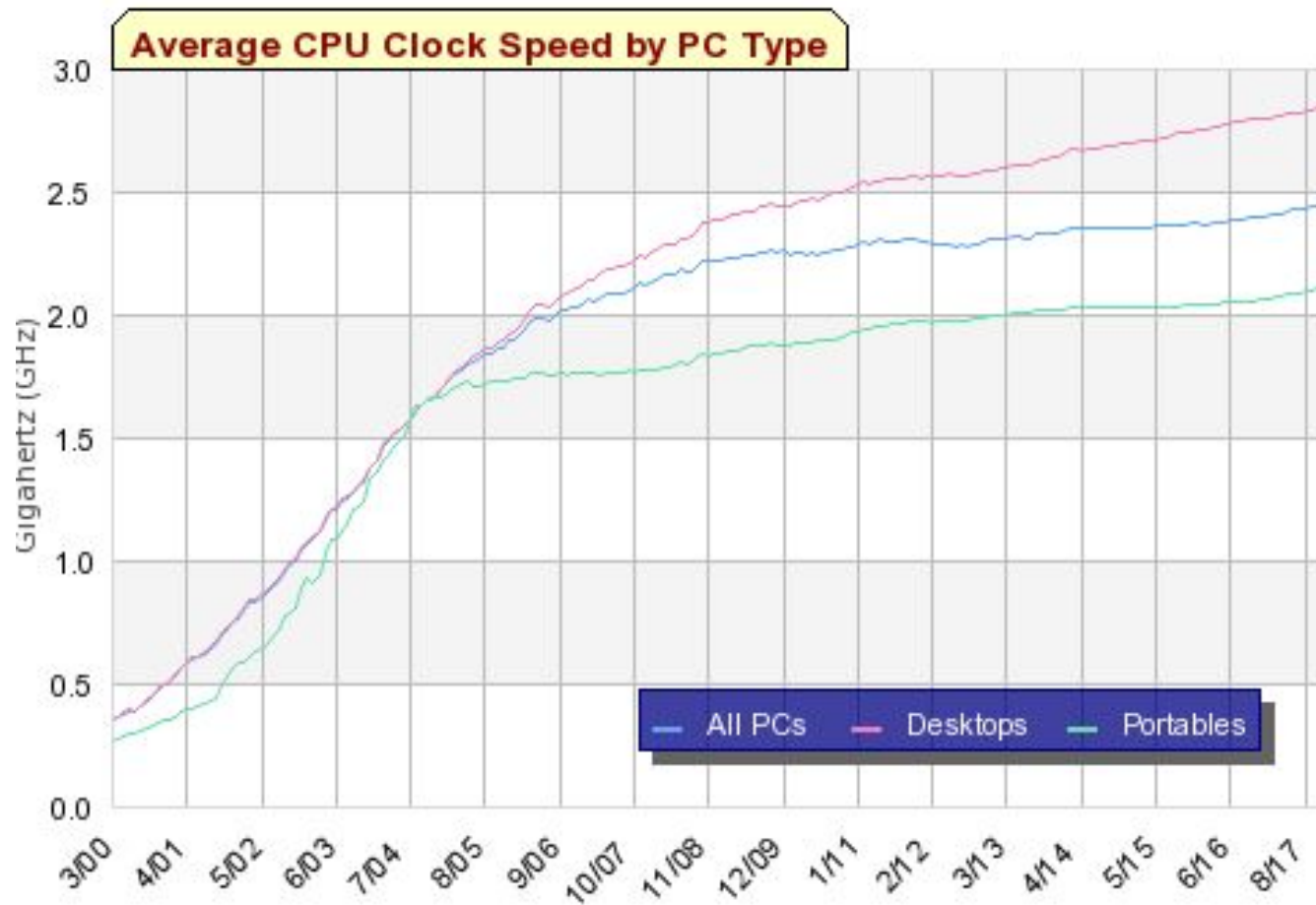
- GCC 7.3 (ou superior, eu uso o 8.1)
 - STD = C++11
- Linux

Objetivo de SuperComputação

Aumentar velocidade de processamento! (Hardware)

- Mais clock
- Memória mais rápida
- Mais núcleos
- Melhor resfriamento

Objetivo de SuperComputação



Objetivo de SuperComputação

Event	Latency	Scaled
1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access (DRAM, from CPU)	120 ns	6 min
Solid-state disk I/O (flash memory)	50–150 µs	2–6 days
Rotational disk I/O	1–10 ms	1–12 months
Internet: San Francisco to New York	40 ms	4 years
Internet: San Francisco to United Kingdom	81 ms	8 years
Internet: San Francisco to Australia	183 ms	19 years
TCP packet retransmit	1–3 s	105–317 years
OS virtualization system reboot	4 s	423 years
SCSI command time-out	30 s	3 millennia
Hardware (HW) virtualization system reboot	40 s	4 millennia
Physical system reboot	5 m	32 millennia

Objetivo de SuperComputação

Aumentar velocidade de processamento! (Software)

Objetivo de SuperComputação

Aumentar velocidade de processamento! (Software)

- Interpretador/JIT/compilador mais rápido
- Algoritmos melhores
- Melhorar organização dos dados

Objetivo de SuperComputação

Notação	Nome	Característica	Exemplo
$O(1)$	constante	independe do tamanho n da entrada	determinar se um número é par ou ímpar; usar uma tabela de dispersão (hash) de tamanho fixo
$O(\log n)$	logarítmica	o problema é dividido em problemas menores	busca binária
$O(n)$	linear	realiza uma operação para cada elemento de entrada	busca sequencial; soma de elementos de um vetor
$O(n \log n)$	log-linear	o problema é dividido em problemas menores e depois junta as soluções	heapsort, quicksort, merge sort
$O(n^2)$	quadrática	itens processados aos pares (geralmente loop aninhado)	bubble sort (pior caso); quick sort (pior caso); selection sort; insertion sort
$O(n^3)$	cúbica		multiplicação de matrizes $n \times n$; todas as triplas de n elementos
$O(n^c), c > 1$	polinomial		caixeiro viajante por programação dinâmica
$O(c^n)$	exponencial	força bruta	todos subconjuntos de n elementos
$O(n!)$	fatorial	força bruta: testa todas as permutações possíveis	caixeiro viajante por força bruta

Objetivo de SuperComputação

Aumentar velocidade de processamento! (Software)

- Interpretador/JIT/compilador mais rápido
- Algoritmos melhores
- Melhorar organização dos dados

Sistemas Hardware-software



Objetivo de SuperComputação

Aumentar velocidade de processamento! (Software)

- Interpretador/JIT/compilador mais rápido
- Algoritmos melhores
- Melhorar organização dos dados

Desafios de Programação



Soluções encontradas

A photograph of the Cray 1 supercomputer, showing its distinctive barrel-shaped modules.	A photograph of the Cray 2 supercomputer, showing its large, modular design.	A photograph of the Cray C-90 supercomputer, showing its large, modular design.	Vector Computers SMP computers
A photograph of the Cosmic cube supercomputer, showing its modular design.	A photograph of the Paragon supercomputer, showing its modular design.	A photograph of the ASCI Red supercomputer, showing its modular design.	Massively Parallel Processors (MPP)
A photograph of a cluster of PCs, showing their modular design.			Cluster Computers Linux PC Clusters (~1995)

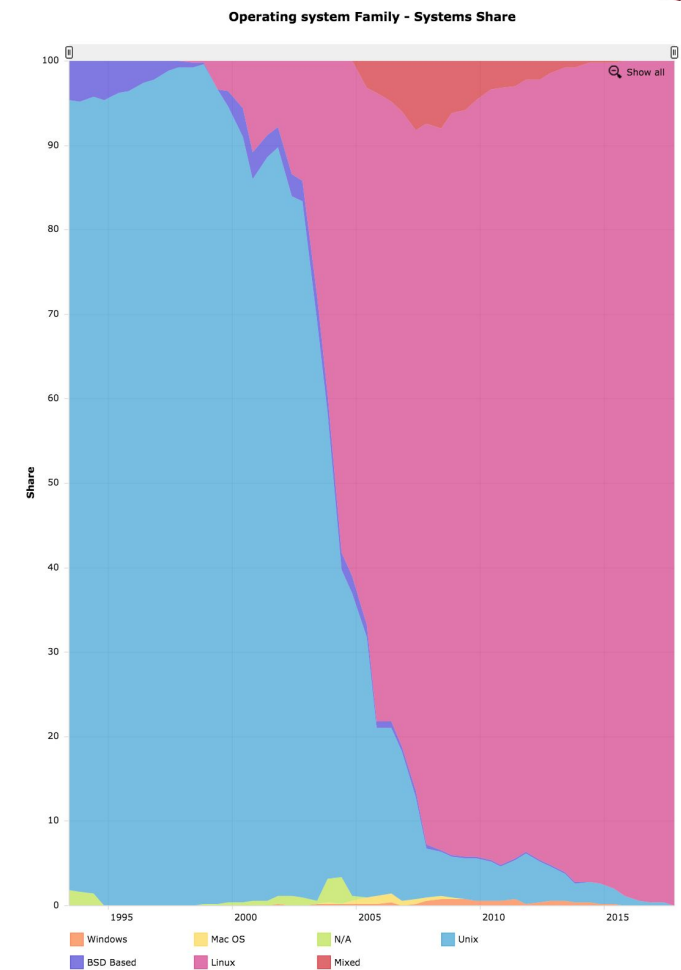
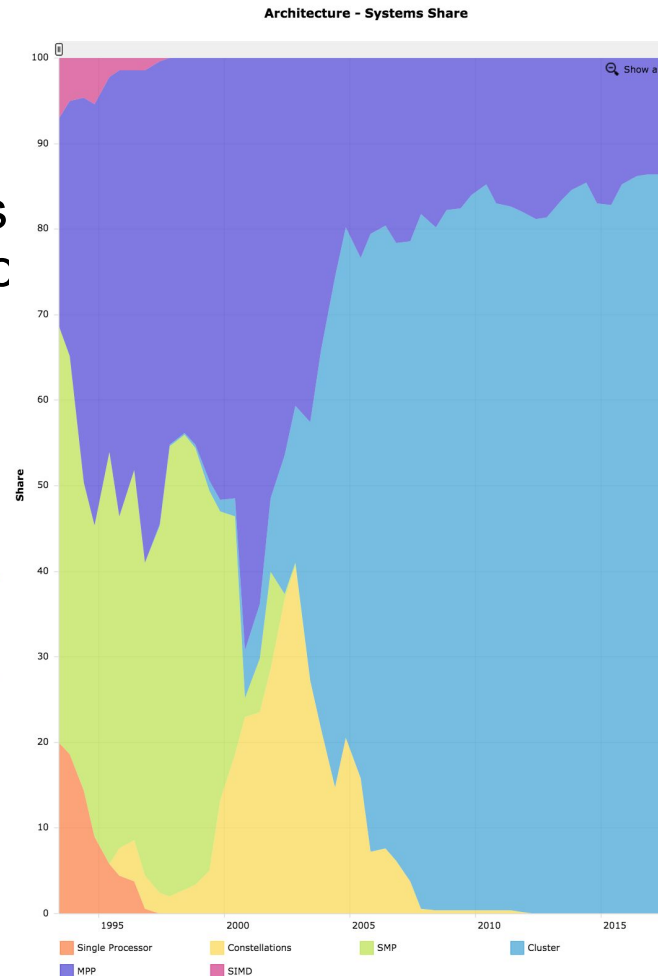
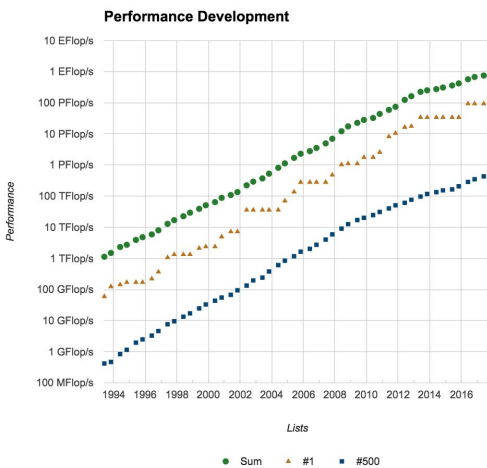


Como programar para estes computadores/arquiteturas?

TOP 500

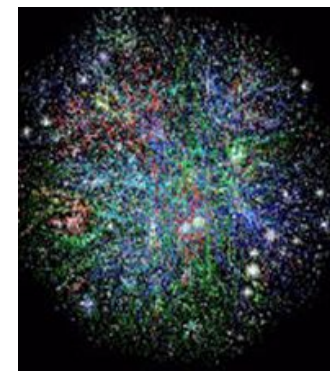
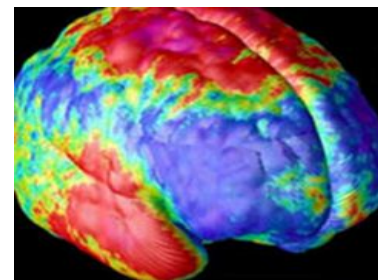
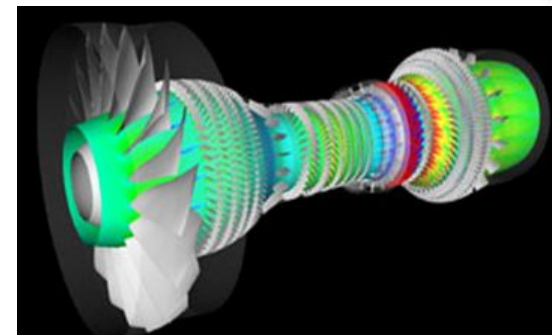
<https://www.top500.org/statistics/overtime/>

- Ranking dos 500 computadores mais poderosos do mundo.



Aplicações de Supercomputação

- Previsão do tempo
- Cálculo de aerodinâmica e car crash
- Análise probabilística
- Modelagem de proteção contra radiação.
- Quebra de senhas por força bruta
- Simulações de testes nucleares 3D
- Simulação de Dinâmica Molecular
- Minização de consumo de combustível por rotas de entrega



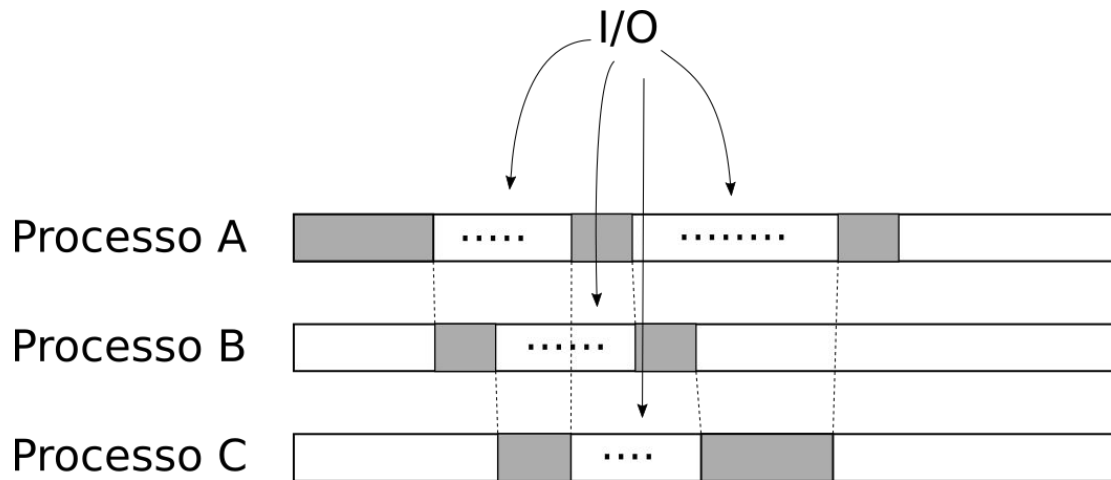
Argnome e
<https://en.wikipedia.org/wiki/Supercomputer>

Conteúdos

- Programação concorrente e sincronização
- Programação paralela em CPUs multi core
- Sistemas distribuídos
- GPGPU

Programação concorrente

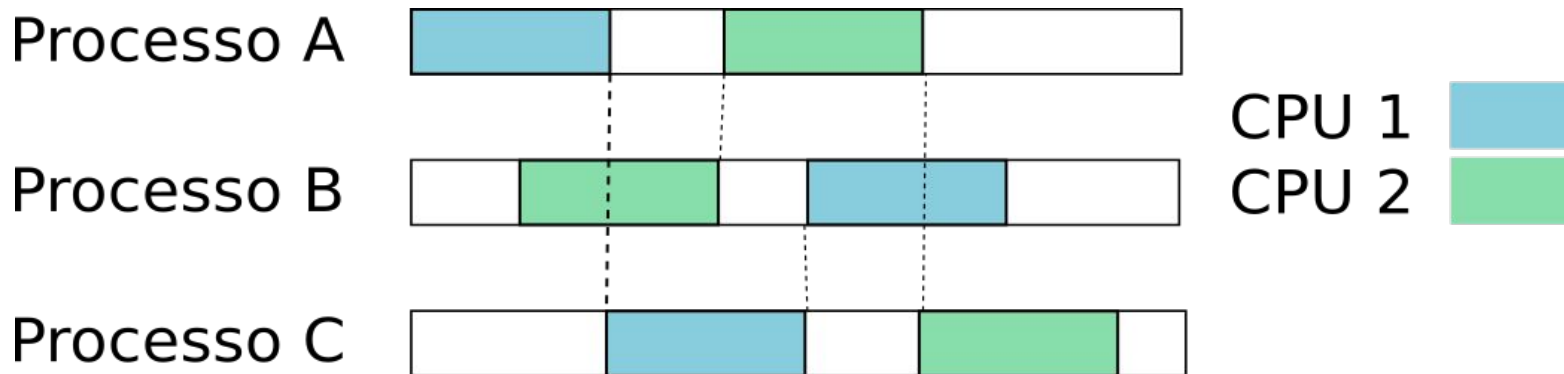
- Tarefas limitadas por entrada e saída



- Estratégia de divisão do problema em tarefas
- Primitivas para sincronizar a execução

Programação Multi core

- Tarefas limitadas por CPU



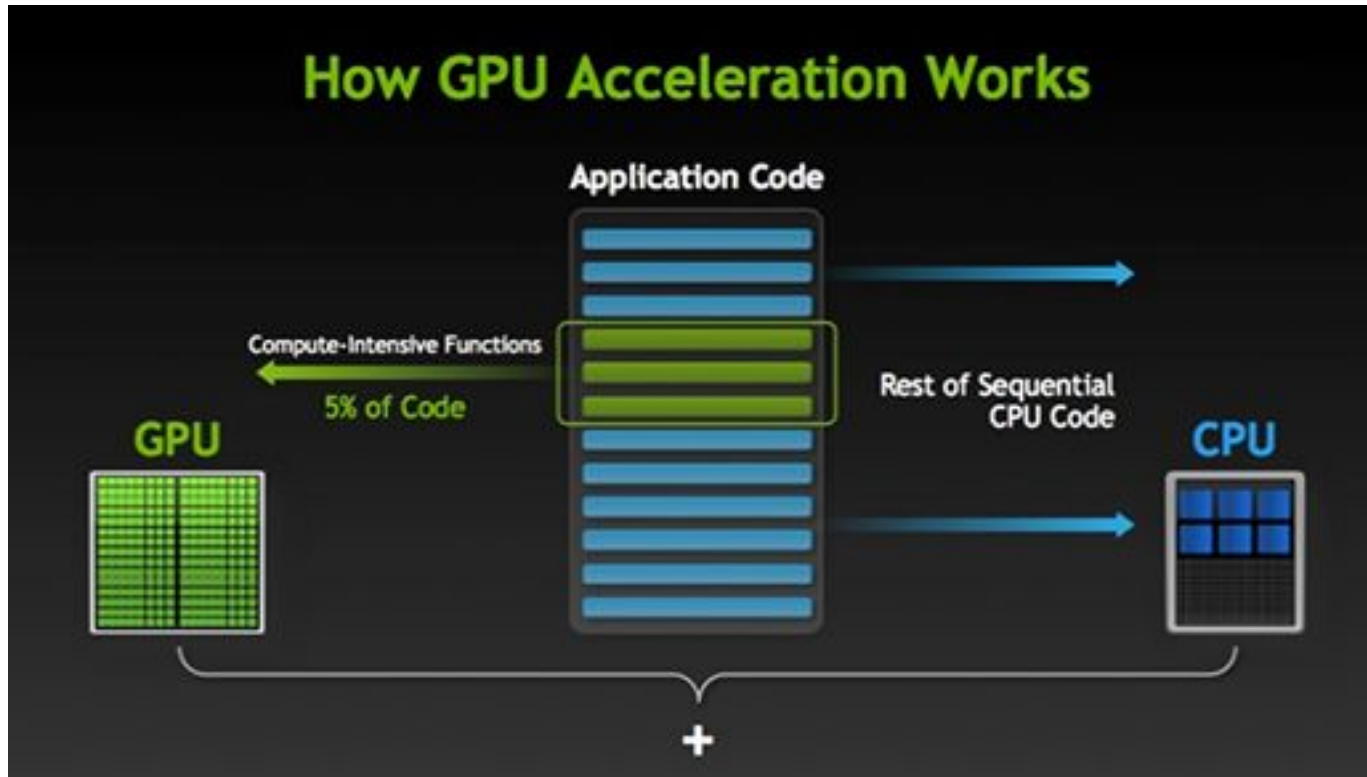
- Divisão em partes independentes
- Modelo fork-join

Sistemas distribuídos



- Divisão de tarefas em clusters
- Passagem de mensagens entre processos/máquinas

GPGPU



- Versão turbinada do modelo multi-core
- Arquitetura completamente diferente

Revolução no acesso a recursos



OPENSIFT

Super Computação sob demanda!

Parte 1 – C++

C++

Nesta aula será apresentado o C++ de uma forma bem simples. Nenhuma instrução ou estratégia será aprofundada. Conforme vocês forem programando os projetos, dúvidas vão surgir e aí sim será o momento de se aprofundar.

C++

C++ é uma linguagem de programação criada no Bell Labs em 1979. Sendo uma das linguagens de programação mais utilizadas no mundo.

C++ é derivado da linguagem C, contudo possui uma série de recursos adicionais que traz grandes vantagens para o desenvolvedor.

Começando

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

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


Hello World

- Um programa “Hello World” em C++ é muito parecido com um em C. Veja o seguinte exemplo:

```
#include <iostream>

int main() {

    std::cout << "Hello World!\n";

}
```

Alguns Pontos Comuns com C

em geral

- 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: `(,)`

Alguns Pontos Comuns com C

em geral

- 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

Algumas Revisões

- Veja se conhece esses recursos.

Rotinas de saltos (jumps)

Break

Interrompe um loop qualquer. Por exemplo:

```
for (int n=10; n>0; n--) {  
    if (n==3) break;  
    std::cout << n << ", ";  
}
```

Continue

Volta para o inicio do bloco do loop. Por exemplo:

```
for (int n=10; n>0; n--) {  
    if (n==3) continue;  
    std::cout << n << ", ";  
}
```

Qual a diferença?

Rotinas de saltos (jumps)

GO TO

Permite saltar para qualquer ponto no código. Por exemplo:

```
int n=10;  
mylabel:  
cout << n << ", ";  
n--;  
if (n>0) goto mylabel;  
cout << "fogo!\n";
```

- Esse recurso é extremamente perigoso se não souber o que está fazendo, use por sua conta e risco.
- Muitas pessoas podem te ofender se encontrarem isso no seu código.
- Em outras palavras, evite realmente usar isso.

Operadores de Endereços

& é o operador de endereço e pode ser lido simplesmente como "endereço de"

* é o operador de de-referência (*dereference*), e pode ser lido como "variável apontada por"

Argumentos passados por valor, por referência e ponteiros


Essa função abaixo ira se comportar como
esperado?

```
void duplicate(int a, int *b) {  
    a *= 2;  
    *b *= 2;  
}
```

O uso de ponteiros e referências pode
economizar transferências de memória.

Funções Inline

Para funções muito simples evita todo o processo de empilhamento e saltos de execução. Em geral o compilador já percebe isso sem você indicar, contudo você pode dar uma ajuda para o compilador.



```
inline string concatenate (const string& a, const string& b)
{
    return a+b;
}
```

Vetores como Parâmetros

Para ter um vetor como parâmetro de uma função, os parâmetros devem ser declarados como o tipo de matriz, mas com colchetes vazios, omitindo o tamanho real da matriz. Por exemplo:

```
void function(int arg[]) { ... }
```

Ponteiros e Vetores

Vetores são criados como um espaço sequencial de memória. A variável (seu identificador) de um vetor aponta para o início dessa posição de memória. Logo um vetor funciona como um ponteiro com o recurso extra de indexação de suas posições.

Isso funciona?

```
int myarray [20];  
int * mypointer;
```

```
mypointer = myarray;
```

Ponteiro Void

Um ponteiro do tipo void não tem tipo definido. Logo não é possível saber o tamanho (qtd. de bytes) e usar o dado apontado diretamente. Para usar o dado você deverá fazer um *casting*. Exemplo:

```
int a = 10;  
void *b = &a;  
int *c = (int *) b;  
int d = *c;
```

Structs

São estrutura em C/C++ que permitem organizar um conjunto de dados de forma única. A sintaxe de structs é a seguinte:

```
struct type_name {  
    member_type1 member_name1;  
    member_type2 member_name2;  
    member_type3 member_name3;  
    .  
    .  
} struct_names;
```

Acessando Structs

Para acessar as structs (não declaradas dinamicamente) você pode usar o símbolo ponto (.).
Por exemplo:

```
struct xpto {  
    int a;  
    float b;  
};  
struct xpto abc;  
abc.a = 20;  
std::cout << abc.a << std::endl;
```

Algumas Novidades

- Verifique agora algumas novidades da linguagem C++ em relação a linguagem C.

Namespaces

- O namespace declara uma região de escopo para os identificadores (tipos, função, variáveis, etc).
- Exemplo do namespace std:

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

- Outra possibilidade

```
using namespace std;
```

```
cout << "Ola";
```

Declarando Namespace

Para declara um namespace use a seguinte sintaxe:

```
namespace identifier
```

```
{
```

```
    named_entities
```

```
}
```

Não se preocupe em criar Namespaces agora, você vai mais usar do que criar num primeiro momento.

Inicializando uma variável

- Tradicionalmente se faz:

```
type identifier = initial_value;
```

- Em C++ é possível se iniciar uma variável:

```
type identifier (initial_value);
```

Exemplo: int x (0);

- Ou ainda:

```
type identifier {initial_value};
```

Exemplo: int x {0};

Iniciando Vetores e 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 corresponda ao número de valores incluídos entre as chaves { }. Exemplo:

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

Alocação de Memória

A memória dinâmica é alocada usando o operador new. O new aloca a quantidade certa de memória se especificando o tipo de dado e seu tamanho para alocar. Ele retorna um ponteiro para o início do novo bloco de memória alocado. Sua sintaxe é:

```
pointer = new type  
pointer = new type [number_of_elements]
```

Exemplo:

```
double * foo;  
foo = new double [5];
```

Desalocando memória

- Use o delete para desalocar a memória alocada com o new. Exemplo:

```
int *pointer1 = new int;
```

```
delete pointer1;
```

```
char *pointer2 = new char[10];
```

```
delete[] pointer2;
```

Números em diferentes bases

- C++ permite o uso de números na base octal (base 8) e números hexadecimais (base 16) como constantes literais.
 - Para literais octais, os dígitos são precedidos por um caractere 0 (zero).
 - Para hexadecimal, eles são precedidos pelos caracteres 0x (zero, x).
- Por exemplo, as seguintes constantes literais são todas equivalentes entre si:

```
75      // decimal
```

```
0113    // octal
```

```
0x4b    // hexadecimal
```

Tipos de dados novos

Principais tipos novos em C++:

- `bool`: que pode ser `true` ou `false`;
- `string`: que armazena textos e possui recursos para tratar o texto

Strings

- Para usar strings você deve colocar o header:

```
#include <string>
```

- Seu uso é bem direto, por exemplo:

```
string texto;
```

```
texto = "exemplo de texto";
```

```
std::cout << texto << std::endl;
```

Input/Output (Streams)

- C++ usa uma abstração conveniente chamada streams para executar entrada e saída de dados. Os dados vão de um lado para o outro de forma contínua.
- Uso com o terminal:

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

Saída de dados

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

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

Entrada de dados

- Use o `std::cin` com dois sinais de menor (`<<`) para indicar que quer capturar os dados do console.

Exemplo:

```
string texto;
```

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

- Outra opção é usando o `getline` que espera o enter (return) para terminar a captura. Exemplo:

```
string texto;
```

```
getline(cin, texto);
```

stringstream

- Um truque que pode ser usado é tratar uma string como um stream de dados usando o stringstream.

Por exemplo:

```
#include <sstream>
```

```
...
```

```
string texto = "11.2";
```

```
float numero;
```

```
stringstream(texto) >> numero;
```

const

const no final de um método significa que este método não pode alterar atributos do objeto, só ler.

Exemplo:

```
class Foo {  
    int x,y;  
    double a,b;  
public:  
    int Bar(int arg1, char &arg2) const {  
        // este método não altera x,y,a ou b.  
        // alterar arg1 ou arg2 não é um problema.  
    }  
};
```

Include Guard

Evita que um arquivo header seja incluído mais de uma vez em uma compilação. A dupla inclusão ou recursão vão levar a um erro de compilação.

Exemplo:

```
#ifndef FOO_H
#define FOO_H

class Foo {
    int x,y;
};

#endif /* FOO_H */
```

Referências

- Livros:

- Hager, G. ; Wellein, G. **Introduction to High Performance Computing for Scientists and Engineers**. 1ª Ed. CRC Press, 2010.

- Artigos:

- Bjarne Stroustrup, “An Overview of the C++ programming language”, THE HANDBOOK OF OBJECT TECHNOLOGY (EDITOR: SABA ZAMIR). CRC PRESS LLC, BOCA RATON. 1999

- Internet:

- <http://www.cplusplus.com/doc/tutorial/>

Referências

Atividade prática:

<https://github.com/Insper/supercomp/wiki>

Entrega: 05/08

Inspire

www.inspire.edub.org