Computação Paralela e Distribuída

Ano lectivo 2023-24

Rascunho - Será reorganizado e actualizado

Tema#01
Introdução

João José da Costa
joao.costa@isptec.co.ao

Março de 2023

Coordenação de Engenharia Informática
Departamento de Engenharias e Tecnologias
Instituto Superior Politécnico de Tecnologias e Ciências

Introdução

Apresentação e Motivação

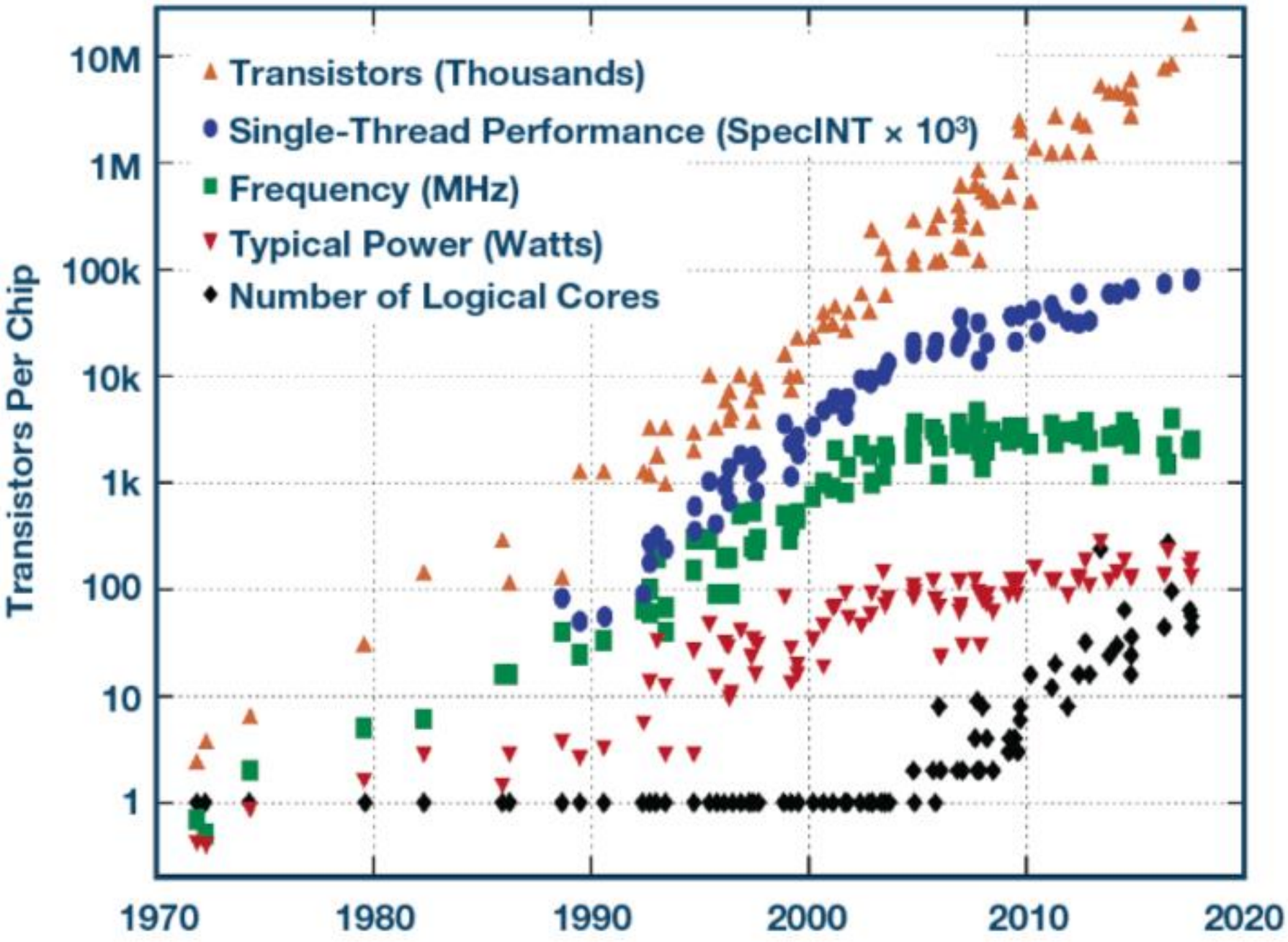
*“Se um **único computador** (processador) consegue resolver **um problema** em **N segundos**, podem **N computadores** (processadores) resolver o **mesmo problema** em **1 segundo?**”*

Motivação

- Modelo de Von Neumann executa instruções sequencialmente.
- Objectivo da computação paralela
  - Reduzir o tempo para obter a solução de problema computacional.

- O mundo real é paralelo por natureza. No entanto...
- A paralelização eficiente de programas é um problema muito difícil!
- A computação paralela tem sido “a próxima grande novidade” nos últimos 40 anos.
- Porquê é importante agora?

Evolução


Transistors (Thousands)
Single-Thread Performance (SpecINT × 10³)
Frequency (MHz)
Typical Power (Watts)
Number of Logical Cores
Transistors Per Chip
10M
1M
100k
10k
1k
100
10
1
1970
1980
1990
2000
2010
2020
Year of Introduction

Analogia (1/2)

- "Contemplar! O carro! diz o funcionário da indústria de carros (computadores). “Ele pode viajar a 30 km/h!”
- “Oh, uau”, diz o consumidor, “essa coisa é fantástica. Posso mover as coisas muito mais rápido do que antes e não preciso recolher cocô de cavalo. Eu quero um!”
- O tempo passa.
- "Contemplar!" diz o funcionário da indústria novamente, “o carro de 60 km/h!”
- "Óptimo!" diz o consumidor. “Eu realmente posso usar isso. A 30 km/h, leva-se o dia todo para chegar à cidade. Isso realmente simplificará enormemente minha vida. Me de me de!"

Analogia (2/2)

- O tempo passa mais uma vez.
- "Contemplar!" diz você-sabe-o-quê, “o carro de 120 km/h!”
- “Oh, eu preciso de um desses. Agora podemos visitar a tia Sadie no outro condado e não ter que passar a noite com seus 42 gatos. Útil! eu compro!”.
- Mais algum tempo.
- "Contemplar!" ele diz: “Dois carros de 100 km/h!”.
- "Diz o que?"

Adaptado de http://perilsofparallel.blogspot.com/

Tendências

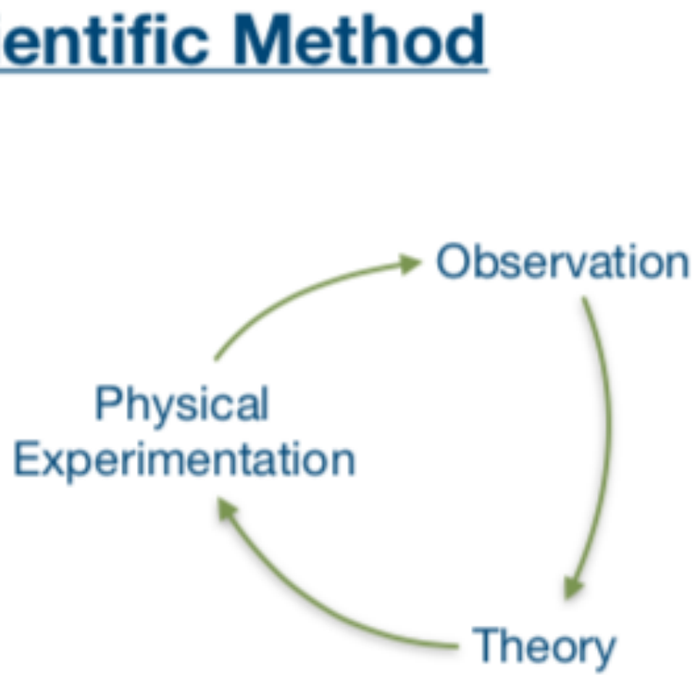
- Multicores.
  - Novos processadores são multicores.
  - ULSI: o que fazer com 100.000 processadores?
- Cluster / Grid / Cloud Computing.
  - grandes clusters feitos de PCs baratos.
  - novos paradigmas: computação em grade e em nuvem

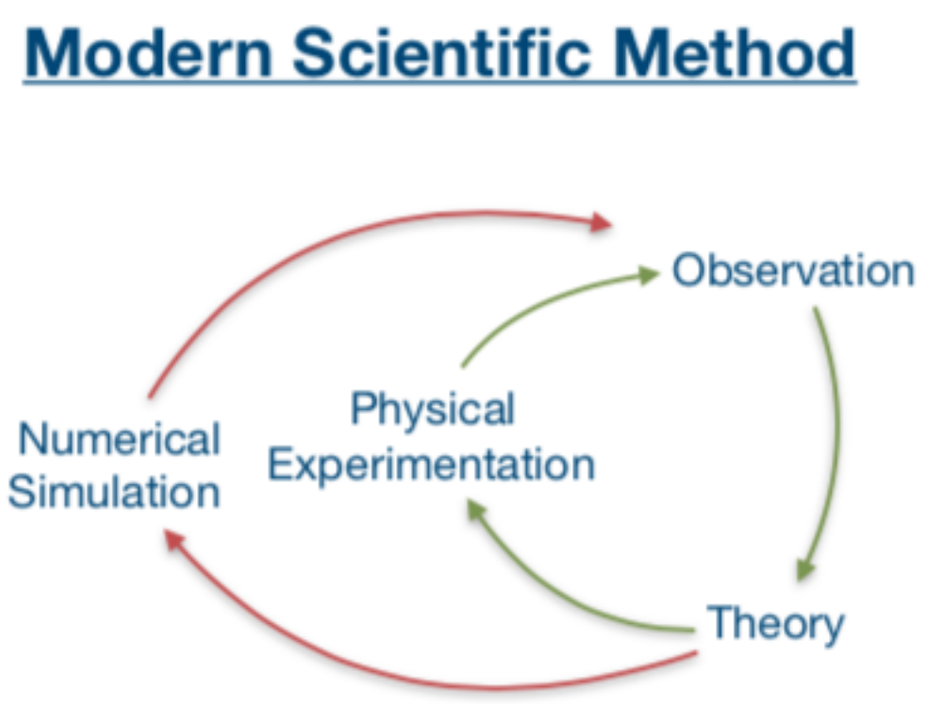
Vantagens

- Velocidade.
  - ganho sobre a execução de processador único
- Economia
  - sistema paralelo mais barato do que um processador mais rápido
  - recursos podem já estar disponíveis
- Escalabilidade.
  - com uma boa arquitectura, fácil de adicionar mais processadores
  - aplicações maiores podem ser executadas com escalas de memória
- Segurança.
  - Redundância de dados

Aplicação

- Tradicionalmente, a programação paralela foi motivada pela resolução/simulação de problemas fundamentais da ciência/engenharia de grande relevância científica e económica, denominados como *Grand Challenge Problems (GCPs)*.

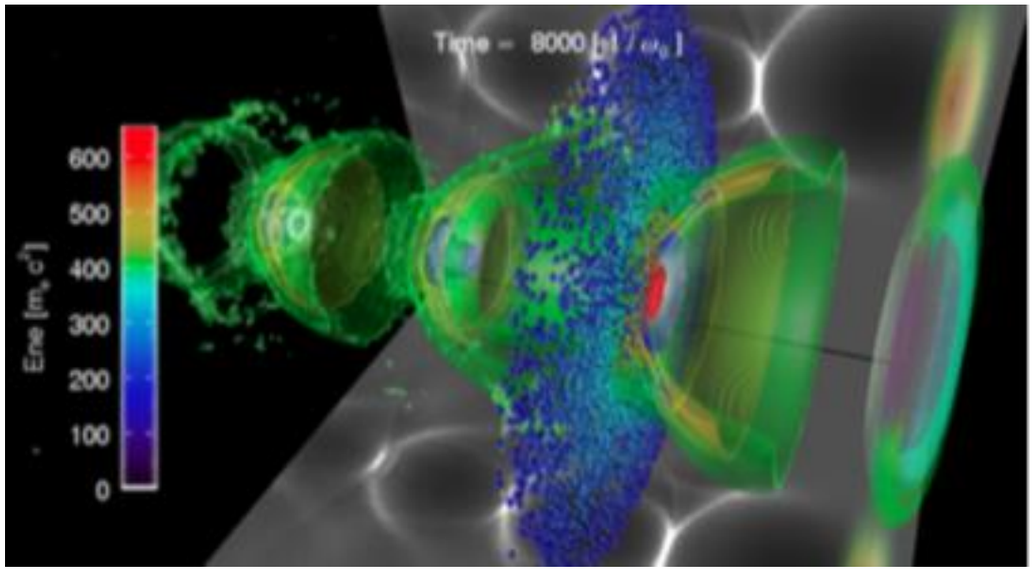




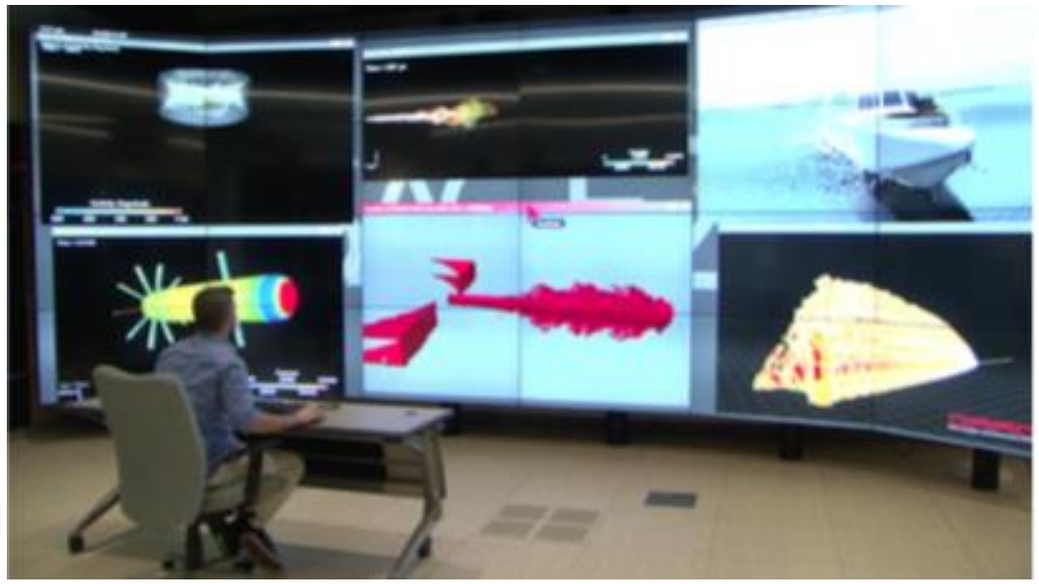
Aplicação (1/5)

- Tipicamente, os GCPs simulam fenómenos que não podem ser medidos por experimentação:
  - Fenómenos climáticos (e.g. movimento das placas tectónicas)
  - Fenómenos físicos (e.g. órbita dos planetas)
  - Fenómenos químicos (e.g. reacções nucleares)
  - Fenómenos biológicos (e.g. genoma humano)
  - Fenómenos geológicos (e.g. actividade sísmica)
  - Componentes mecânicos (e.g. aerodinâmica/resistência de materiais em naves espaciais)
  - Circuitos electrónicos (e.g. verificação de placas de computador)
  - …

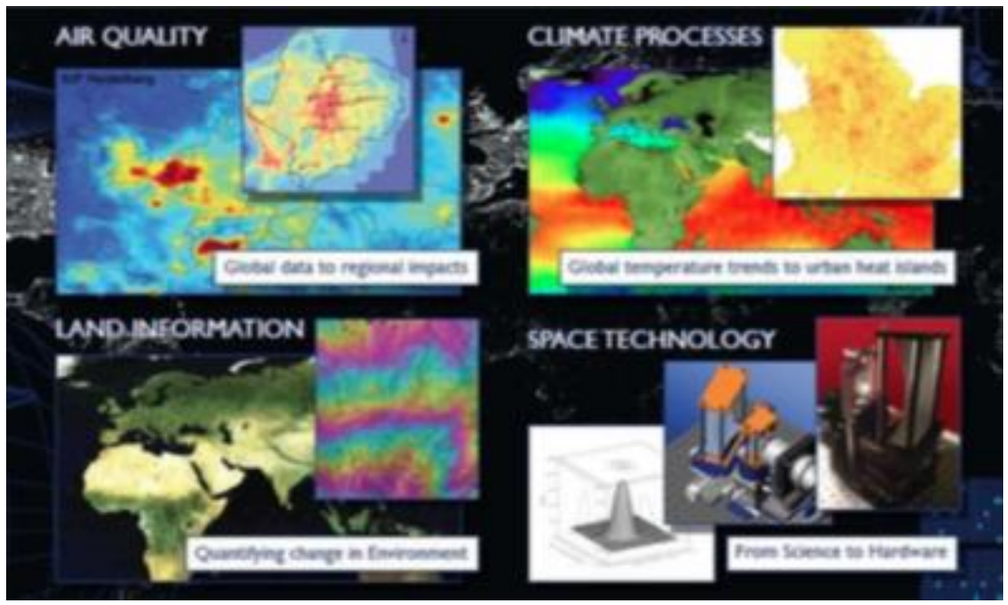
Aplicação (2/5)



Aplicação científica



Aplicação militar



Observação da terra

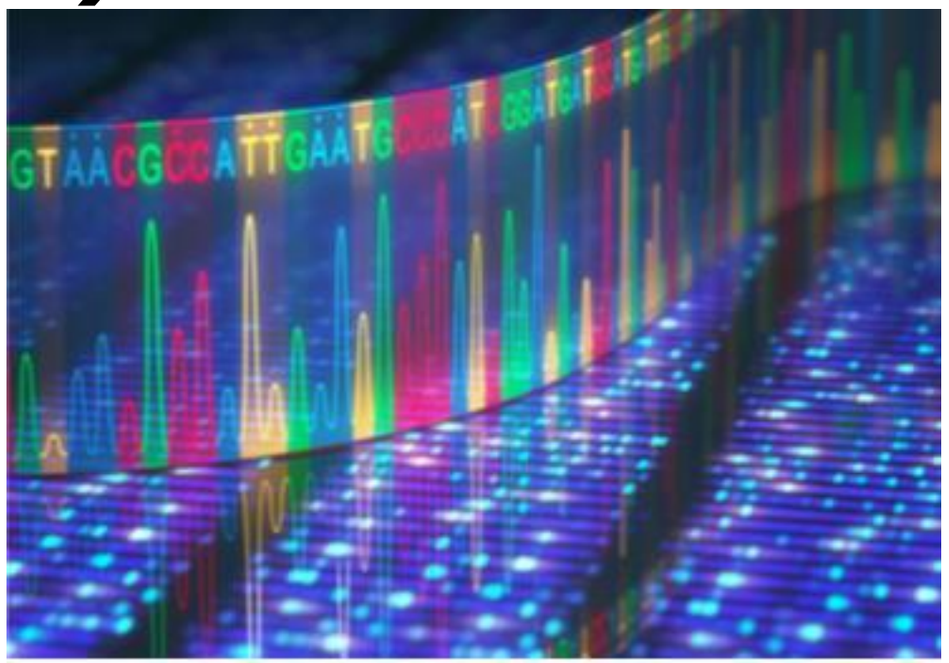
- **Muito caro**
- **Muito difícil**
- **Muito perigoso**
- **Muito controverso**
- **Muito proibido**

Aplicação (3/5)

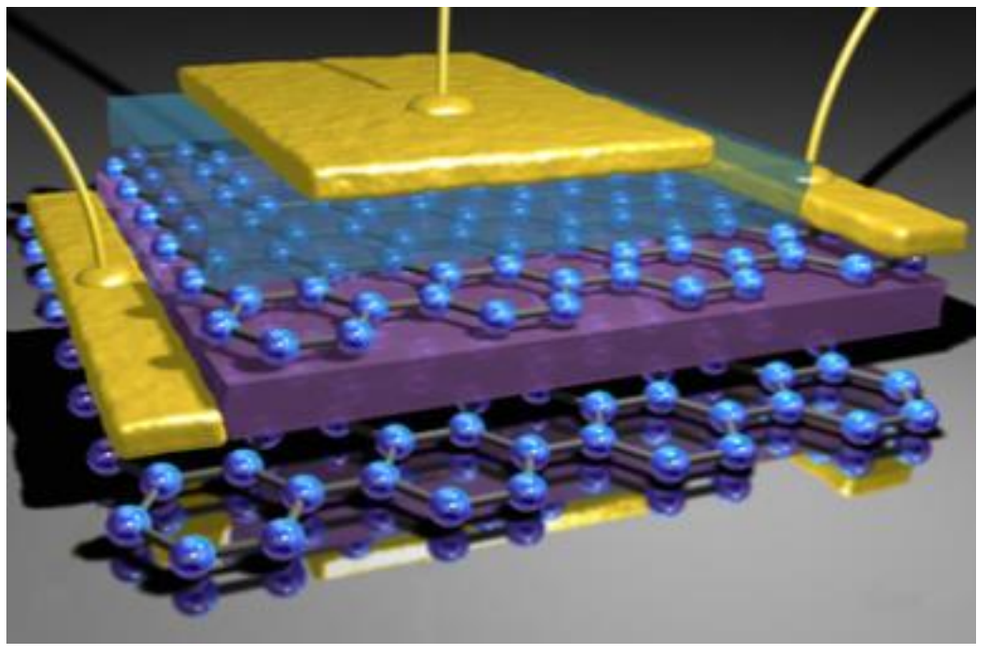
- Actualmente, as aplicações que exigem o desenvolvimento de computadores cada vez mais rápidos estão por todo o lado. Estas aplicações ou requerem um grande poder de computação ou requerem o processamento de grandes quantidades de informação. Alguns exemplos são:
  - Bases de dados paralelas
  - Mineração de dados (*data mining*)
  - Serviços de procura baseados na web
  - Serviços associados a tecnologias multimédia e telecomunicações
  - Computação gráfica e realidade virtual
  - Diagnóstico médico assistido por computador
  - Gestão de grandes industrias/corporações
  - …

Aplicação (4/5)

Serviços financeiros



Ciências da vida



Novas tecnologias

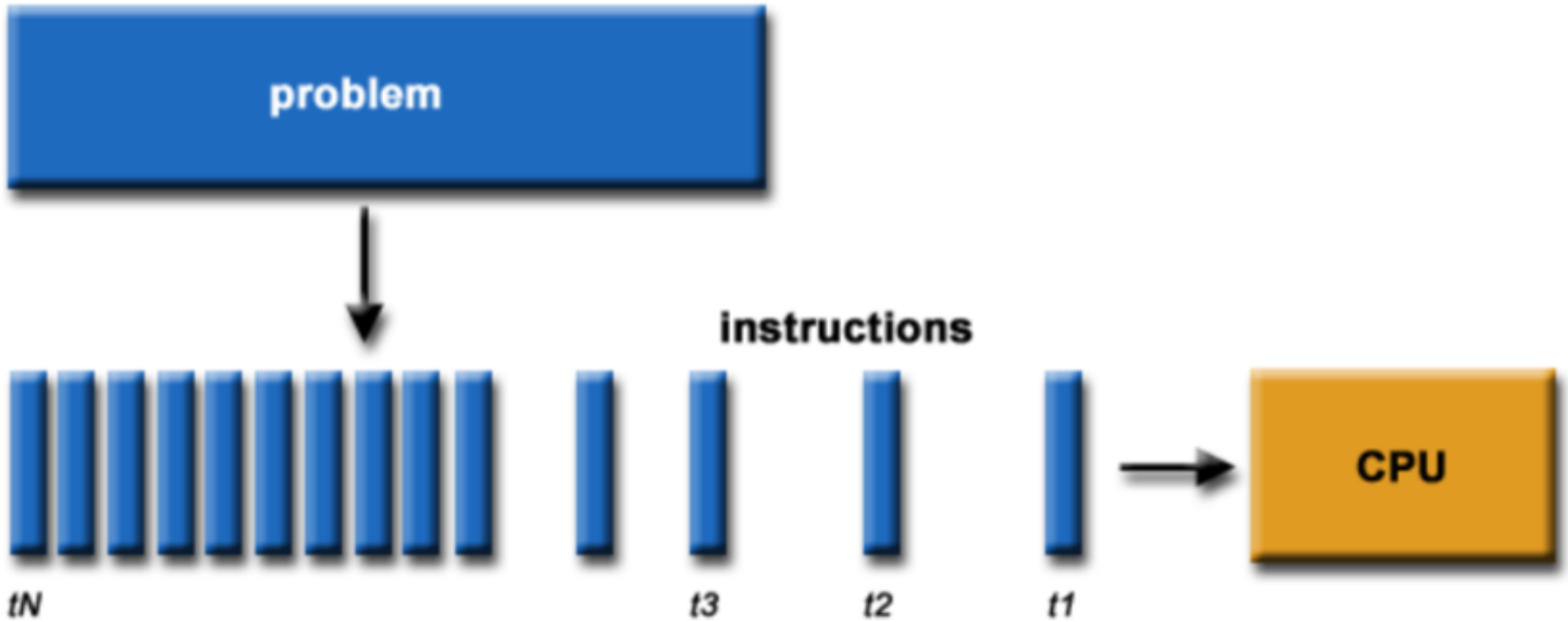
Óleo, Gás, energias renováveis

Aplicação (5/5)

Ciências de dados

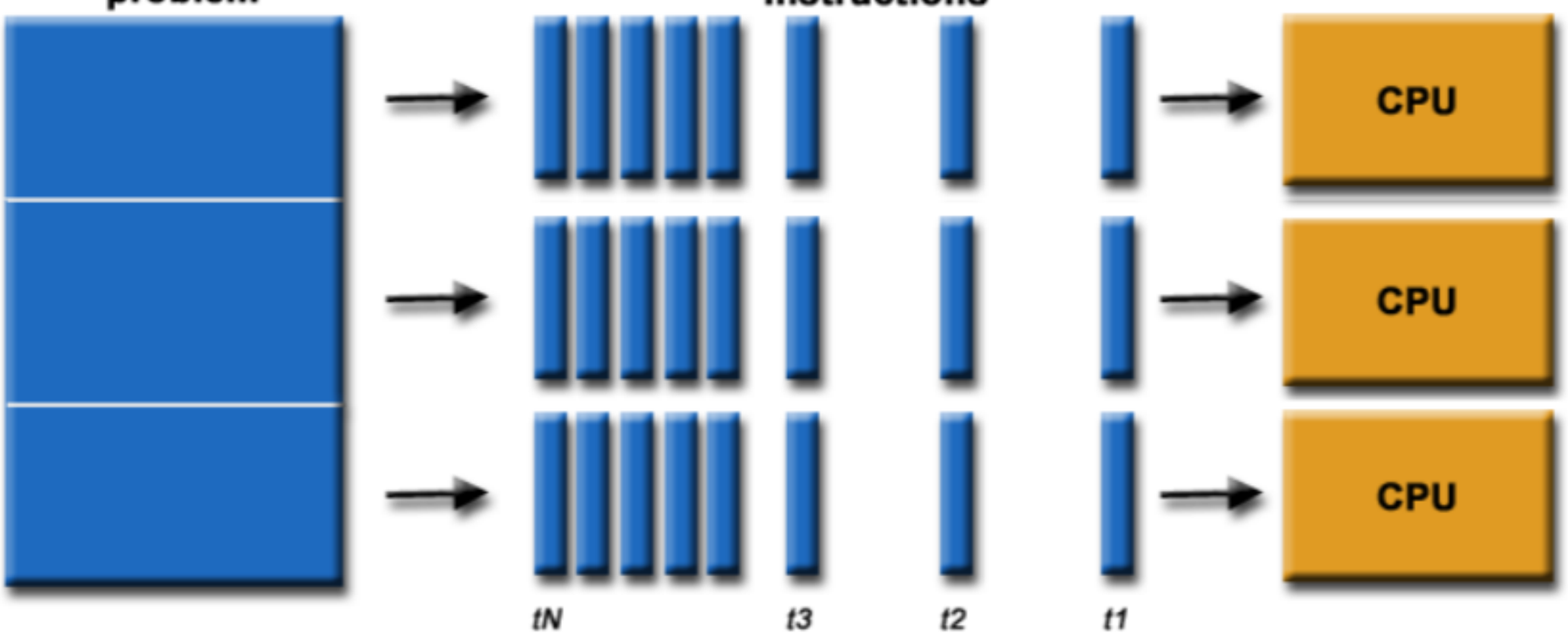
Programação serial

- Um programa é considerado programação serial quando este é visto como uma série de instruções sequenciais que devem ser executadas num único processador.



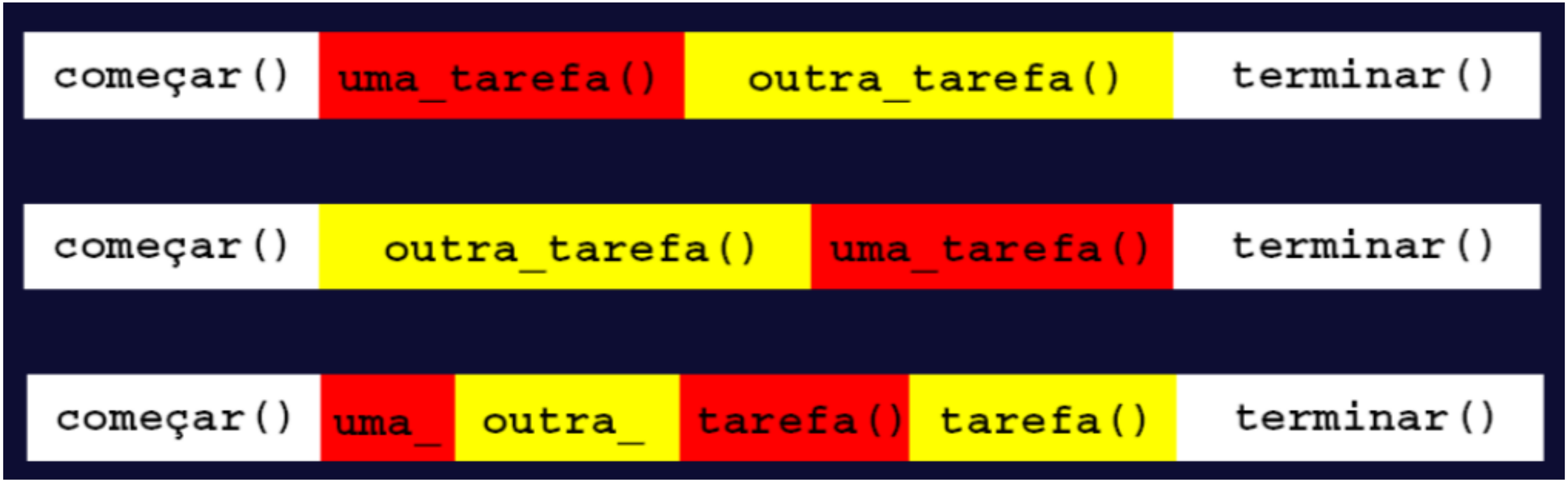
Programação paralela

- Um programa é considerado programação paralela quando este é visto como um conjunto de partes que podem ser resolvidas **concorrentemente**. Cada parte é igualmente constituída por uma série de instruções sequenciais, mas que no seu conjunto podem ser executadas simultaneamente em vários processadores.



Paralelismo potencial

- Concorrência ou paralelismo potencial diz-se quando um programa possui **tarefas** (partes contíguas do programa) que podem ser executadas em qualquer ordem sem alterar o resultado final.



Paralelismo

- Paralelismo diz-se quando as tarefas de um programa são executadas em simultâneo em mais de um processador.

Paralelismo implícito

- O paralelismo diz-se implícito quando cabe ao **compilador** e ao **sistema** de execução:
  - **Detectar** o paralelismo potencial do programa.
  - **Atribuir** as tarefas para execução em paralelo.
  - **Controlar** e **sincronizar** toda a execução.

- Vantagens e inconvenientes:
  - ✶Liberta o programador dos detalhes da execução paralela. (+)
  - ✶Solução mais geral e mais flexível. (+)
  - ✶Difícil conseguir-se uma solução eficiente para todos os casos. (–)

Paralelismo explícito

- O paralelismo diz-se explícito quando cabe ao programador:
  - **Anotar** as tarefas para execução em paralelo.
  - **Atribuir** (possivelmente) as tarefas aos processadores.
  - **Controlar** a execução indicando os pontos de sincronização.
  - **Conhecer a arquitectura dos computadores** de forma a conseguir o máximo desempenho (aumentar localidade, diminuir comunicação, etc).

- Vantagens e inconvenientes:
  - ✶Programadores experientes produzem soluções muito eficientes para problemas específicos. (+)
  - ✶O programador é o responsável por todos os detalhes da execução (debugging pode ser penoso). (–)
  - ✶Pouco portável entre diferentes arquitecturas. (–)

Computação paralela

- De uma forma simples, a computação paralela pode ser definida como o uso simultâneo de vários recursos computacionais de forma a reduzir o tempo necessário para resolver um determinado problema. Esses recursos computacionais podem incluir:
  - Um único computador com múltiplos processadores.
  - Um número arbitrário de computadores ligados por rede.
  - A combinação de ambos.

Programação Paralela

- Apesar das arquitecturas paralelas serem actualmente uma realidade, a programação paralela continua a ser uma tarefa complexa.

- Para além de depender da disponibilidade de ferramentas/ambientes de programação adequados para memória partilhada/distribuída, debate-se com uma série de problemas não existentes em programação sequencial

Programação Paralela

- **Principais problemas:**
  - **Concorrência**: identificar as partes da computação que podem ser executadas em simultâneo.

  - **Comunicação** e **Sincronização**: desenhar o fluxo de informação de modo a que a computação possa ser executada em simultâneo pelos diversos processadores evitando situações de *deadlock* e *race conditions*.

  - **Balanceamento de Carga** e **Escalonamento**: distribuir de forma equilibrada e eficiente as diferentes partes da computação pelos diversos processadores de modo a ter os processadores maioritariamente ocupados durante toda a execução.

Limites de desempenho

Factores que limitam o desempenho

- Código Serial: existem partes do código que são inerentemente seriais (p.e. iniciar/terminar a computação).
- Concorrência: o número de tarefas pode ser escasso e/ou de difícil definição. Comunicação: existe sempre um custo associado à troca de informação e enquanto as tarefas processam essa informação não contribuem para a computação.
- Sincronização: a partilha de dados entre as várias tarefas pode levar a problemas de contenção no acesso à memória e enquanto as tarefas ficam à espera de sincronizar não contribuem para a computação.

Limites de desempenho

Factores que limitam o desempenho

- Granularidade: o número e o tamanho das tarefas é importante porque o tempo que demoram a ser executadas tem de compensar os custos da execução em paralelo (e.g. custos de criação, comunicação e sincronização).
- Balanceamento de Carga: ter os processadores maioritariamente ocupados durante toda a execução é decisivo para o desempenho global do sistema.

Principais modelos

Principais Modelos de Programação Paralela

- Programação em Memória Partilhada
  - ✓ Programação usando processos ou threads.
  - ✓ Decomposição do domínio ou funcional com granularidade fina, média ou grossa.
  - ✓ Comunicação através de memória partilhada.
  - ✓ Sincronização através de mecanismos de exclusão mútua.
- Programação em Memória Distribuída
  - ✓ Programação usando troca de mensagens.
  - ✓ Decomposição do domínio com granularidade grossa.
  - ✓ Comunicação e sincronização por troca de mensagens.

Objectivos

- Aprender diferentes **arquitecturas** paralelas e **modelos** de programação
- Entender as **dificuldades** sobre computação paralela e distribuída
- Estudar **metodologias** para programação paralela eficiente
- **Praticar a programação** em arquitectura de memória partilhada e distribuída
- **Analisar algoritmos** em diferentes áreas de aplicação

Avaliação (1/2)

- Componente teórica: 60%
  - Provas escritas
- Componente prática: 40%
  - Projecto em grupo
- Nota
  - PP1 = (Projecto OpenMP)*0,4 + (Prova escrita)*0,6
  - PP2 = (Projecto MPI)*0,4 + (Prova escrita)*0,6
  - Exame = (Projecto)*0,4 + (Prova escrita)*0,6

Avaliação (2/2)

- Um projecto, com 2 componentes:
  - **Programação com OpenMP**, prazo de entrega para 6/4/2024
  - **Programação com MPI**, prazo de entrega para 18/5/2023
  - **N.B.:** (i) grupo de 3 estudantes, (ii) avaliação consiste em demo/discussão e (iii) nota individual determinada na avaliação.

- Nota Final do Projecto (componente prática)
  - Programação com OpenMP: 40%
  - Programação com MPI: 60%
  - **Obs:** nota a ser usada para o cálculo da nota do exame final.

Tecnologias

- **Base**
  - Linux, GCC, GDB, MAKE, GIT
- **Editor**
  - VI, VSCode, etc.
- **Principais biblioteca**
  - OpenMP
  - MPI

Bibliografia

- Parallel Programming in C with MPI and OpenMP, Michael Quinn, McGraw-Hill
- Parallel Programming, B. Wilkinson and M. Allen, Prentice Hall
- Designing and Building Parallel Programs: Concepts and tools for Parallel Software Engineering, Ian Foster, Addison Wesley
- Artigos científicos e outras referências serão dadas ao longo do semestre

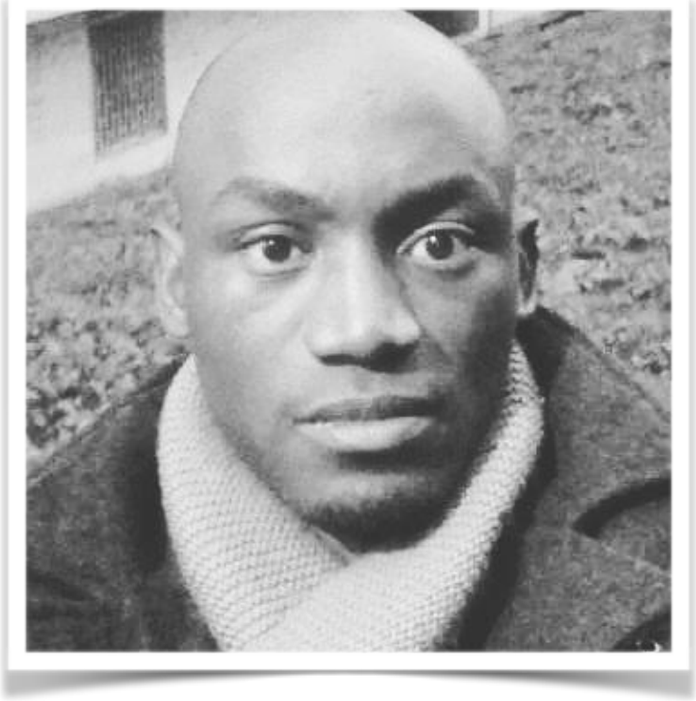
Programa (1/2)

- Visão geral
  - Arquitectura de computadores
  - Análise de algoritmos
- Arquitectura paralela - memória partilhada e distribuída
- Análise de programa - paralelismo de dado e funcional
- Metodologia
  - Particionamento, comunicação, aglomeração e mapeamento.

Programa (2/2)

- Análise de programas paralelo
  - Métricas de desempenho
  - Debugging
  - Limitações fundamentais
- Estudos de casos
  - Ordenação, pesquisa e optimização
  - Algoritmos de grafos
  - Numérico: multiplicação de matrizes, …

Docente



João José da Costa, Licenciado na FC-UAN e Mestrado no IST-ULisboa.

Áreas de interesse: Arquitectura de Computadores; Sistemas Operativos; Virtualização; Computação Utilitária/Nuvem; Armazenamento na Nuvem; Processamento de Dados Distribuídos; Replicação de Dados; Computação Móvel e Ubíqua.

Aulas teórico-práticas + aulas práticas + Laboratórios

Introdução

Introdução à programação e arquiteturas paralelas

Objectivos

- Fornecer exemplo ilustrativo de oportunidades de paralelização;
- Introduzir a noção de speedup e overheads.
- Apresentar uma visão geral das áreas de aplicação e dos sistemas paralelos.

Assuntos

- Oportunidades para paralelismo (exemplo simples)
- Speedup e Overheads
- Classificação de arquitectura paralela
- Modelos de programação

Oportunidades (1/2)

```
x = initX(A, B);
y = initY(A, B);
z = initZ(A, B);

for(i = 0; i < N_ENTRIES; i++)
  x[i] = compX(y[i], z[i]);
```

Como melhorar o desempenho ????

```
for(i = 1; i < N_ENTRIES; i++){
  x[i] = solveX(x[i-1]);
  z[i] = x[i] + y[i];
}

finalize1(&x, &y, &z);
finalize2(&x, &y, &z);
finalize3(&x, &y, &z);
```

Oportunidades (2/2)

```
x = initX(A, B);
y = initY(A, B);
z = initZ(A, B);
```

Diferentes rotinas executadas em cada unidade de processamento.

Paralelismo funcional

```
for(i = 0; i < N_ENTRIES; i++)
  x[i] = compX(y[i], z[i]);
```

Paralelismo de dados

Mesma rotina sobre diferentes dados em execução em cada unidade de processamento.

```
for(i = 1; i < N_ENTRIES; i++){
  x[i] = solveX(x[i-1]);
  z[i] = x[i] + y[i];
}
```

Pipelining

```
finalize1(&x, &y, &z);
finalize2(&x, &y, &z);
finalize3(&x, &y, &z);
```

Não é bom?! Justifique.

Assuntos

- Oportunidades para paralelismo (exemplo simples)
- Speedup e Overheads
- Classificação de arquitectura paralela
- Modelos de programação

Quão rápido?

- Principal objectivo: reduzir o tempo de execução.
- Speedup:

$$S = \frac{T_{\text{serial}}}{T_{\text{parallel}}}$$

- Speedup ideal com p processadores? p .
- Speedup esperado? < p .
- Não podemos obter speedup superlinear, S > P ? Sim.
  - Eficiência aumentada em acesso a memória (mais registadores, cache)
  - Alguns problemas específicos (por exemplo, pesquisa)

Limitações

Limitações para o speedup ideal (overheads):

- Transferências de dados (ou tipicamente, comunicação entre as tarefas)
- Inicialização/término de tarefa
- Balanceamento de carga
- Porção inerentemente sequencial da computação

Efeitos

Trabalho distribuído perfeitamente entre os processadores = tempo de execução serial dividido por p.

- Efeitos da fracção sequencial

$$f = \frac{T_s}{T_{serial}}$$

$$T_{parallel} = f.T_{serial} + (1 - f).\frac{T_{serial}}{p}$$

Lei de Amdahl

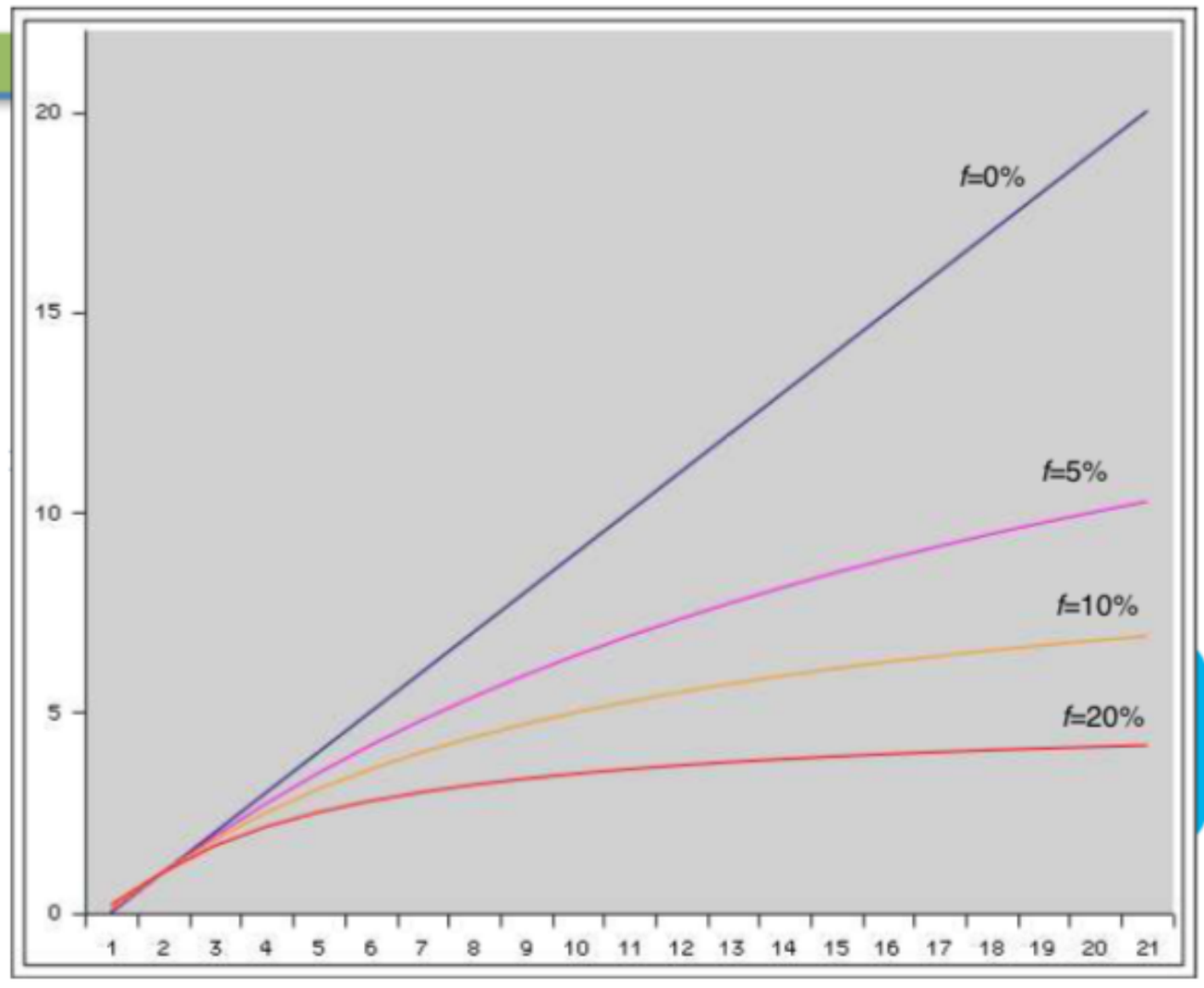
$$S(p, f) = \frac{T_{serial}}{T_{parallel}} = \frac{1}{f + \frac{1-f}{p}}$$

Efeitos

T_s T_{serial}

$T_{parallel}$

$T_{parallel}$



Lei de Amdahl

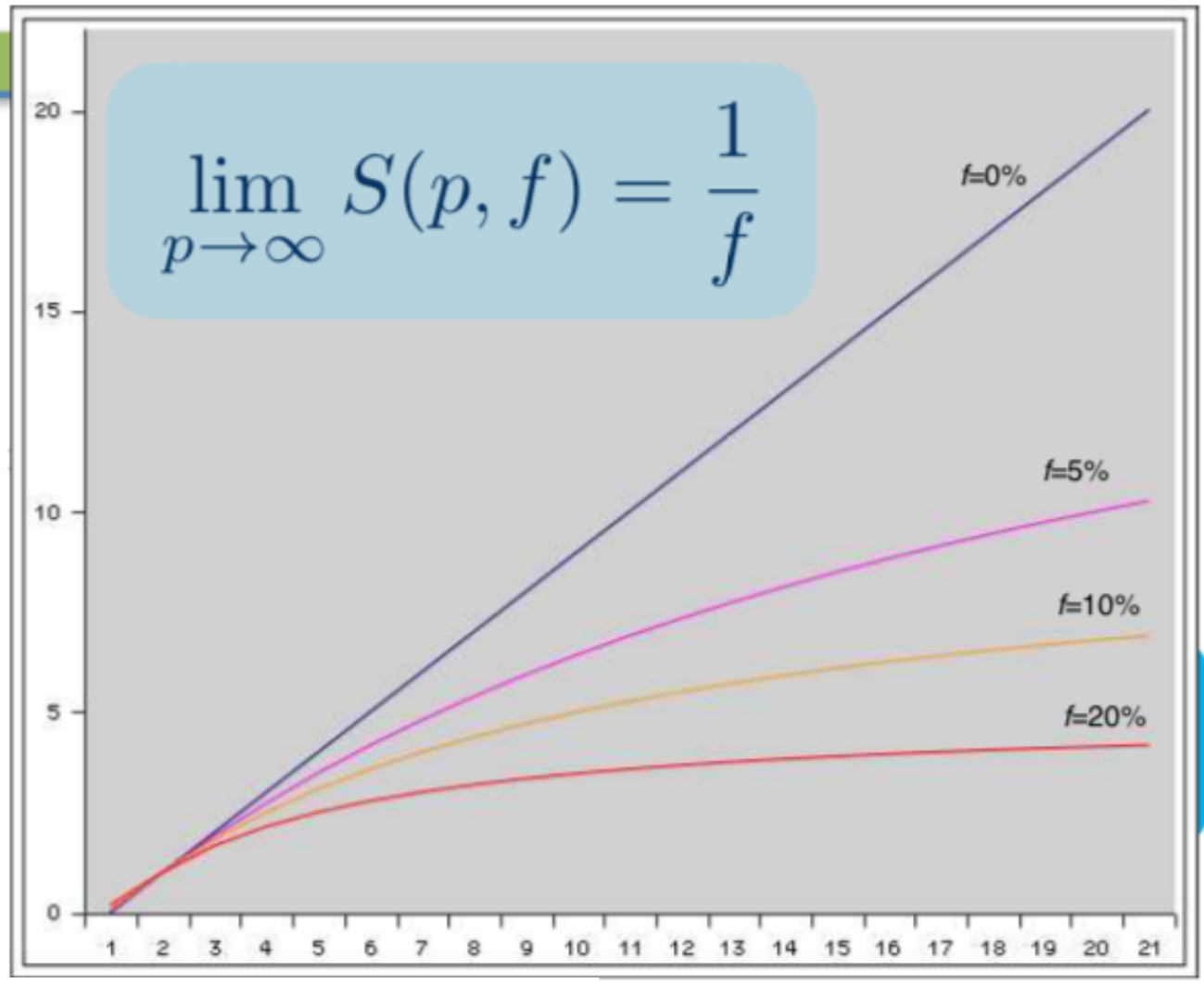
$$S(p, f) = \frac{T_{\text{serial}}}{T_{\text{parallel}}} = \frac{1}{f + \frac{1-f}{p}}$$

Efeitos

T_s T_{serial}

T_{parallel}

T_{parallel}



Lei de Amdahl

$$S(p,f) = \frac{T_{\text{serial}}}{T_{\text{parallel}}} = \frac{1}{f + \frac{1-f}{p}}$$

Dificuldades

- Desenvolvimento de algoritmo é difícil
  - Definir e coordernar tarefas concorrentes.
- Programação de software mais complexo
  - Directivas paralelas de baixo nível
  - Debug significativamente mais difíil
  - Falta de modelos e ambientes de programação
- Rápidas mudanças na arquitectura de sistema de computador
  - Algoritmo paralelo pode não ser eficiente para a próxima geração de computadores parelalos

Dados partilhados

Acesso a dados partilhados

- Uma dependência de dados é uma ordenação em um par de operações de memória que devem ser preservadas para manter a correcção.

- Existe uma condição de corrida quando o resultado de uma execução depende do tempo de dois ou mais eventos.

- A sincronização é usada para sequenciar o controlo entre threads ou para sequenciar acessos a dados em código paralelo.

Novos problemas

- Deadlock: uma tarefa a aguardar por uma condição que nunca é satisfeita
  - Tarefa A solicita acesso ao recurso X, que é retido pela tarefa B que foi bloqueada a aguardar que a tarefa A liberte o recurso Y.

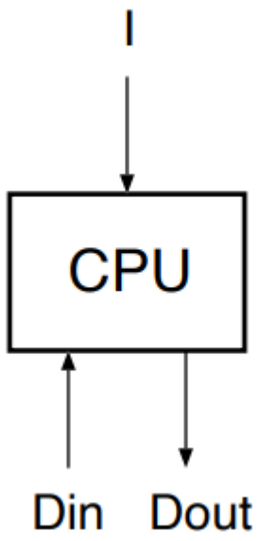
- Livelock: semelhante ao Deadlock, excepto que as tarefas envolvidas no livelock estão em execução, mas nenhuma progride.
  - Duas tarefas a executarem o mesmo contador, uma a diminuir o contador e espera que chegue a 0, a outra a aumentar a espera que chegue a N..

Classificação

Classificação de Arquitectura de Computadores

Taxonomia de Flynn

- Single Instruction, Single Data (SISD)
  - Caso de uni-processador

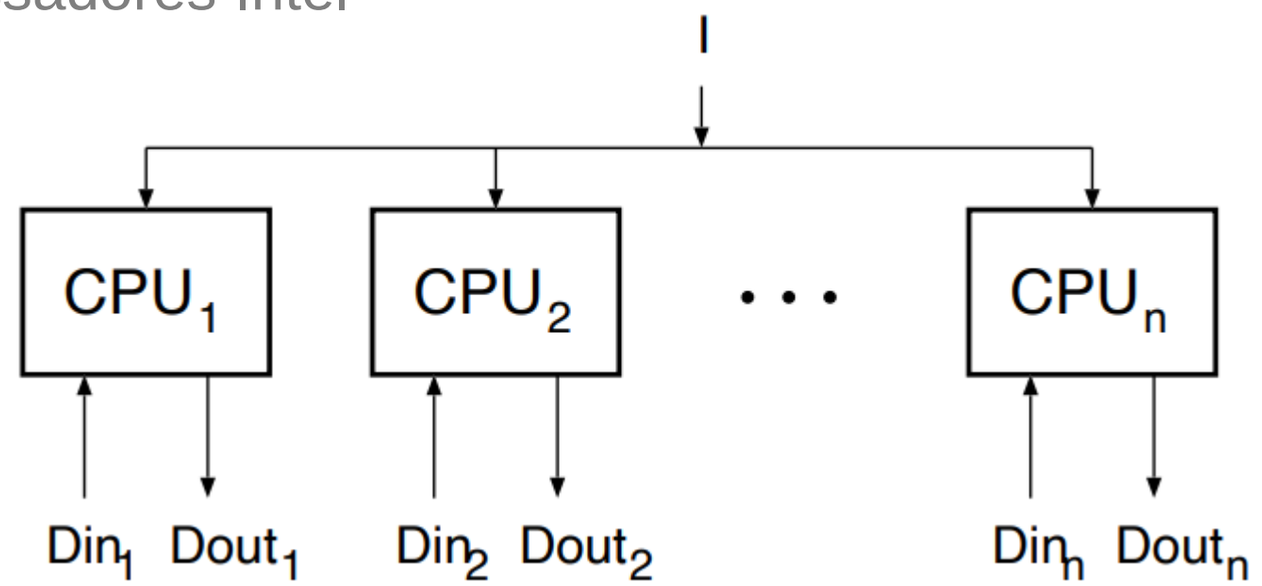


Classificação

Classificação de Arquitectura de Computadores

Taxonomia de Flynn

- Single Instruction, Single Data (SISD)
  - A mesma instrução executada em todos os elementos de processamento, porém cada um em um conjunto diferente de dados
  - Processadores vectoriais (por exemplo, GPU), Instruções SSE em processadores Intel



Classificação

Classificação de Arquitectura de Computadores

Taxonomia de Flynn

- Extensões SIMD multimídia
  - A variação mais amplamente utilizada do SIMD é, actualmente, encontrada em quase todos os microprocessadores.
  - Base de instruções MMX/SSE/AVX adicionadas para melhorar o desempenho de programas multimídia
  - Uma única ALU ampla é particionada em muitas ALUs menores que operam em paralelo

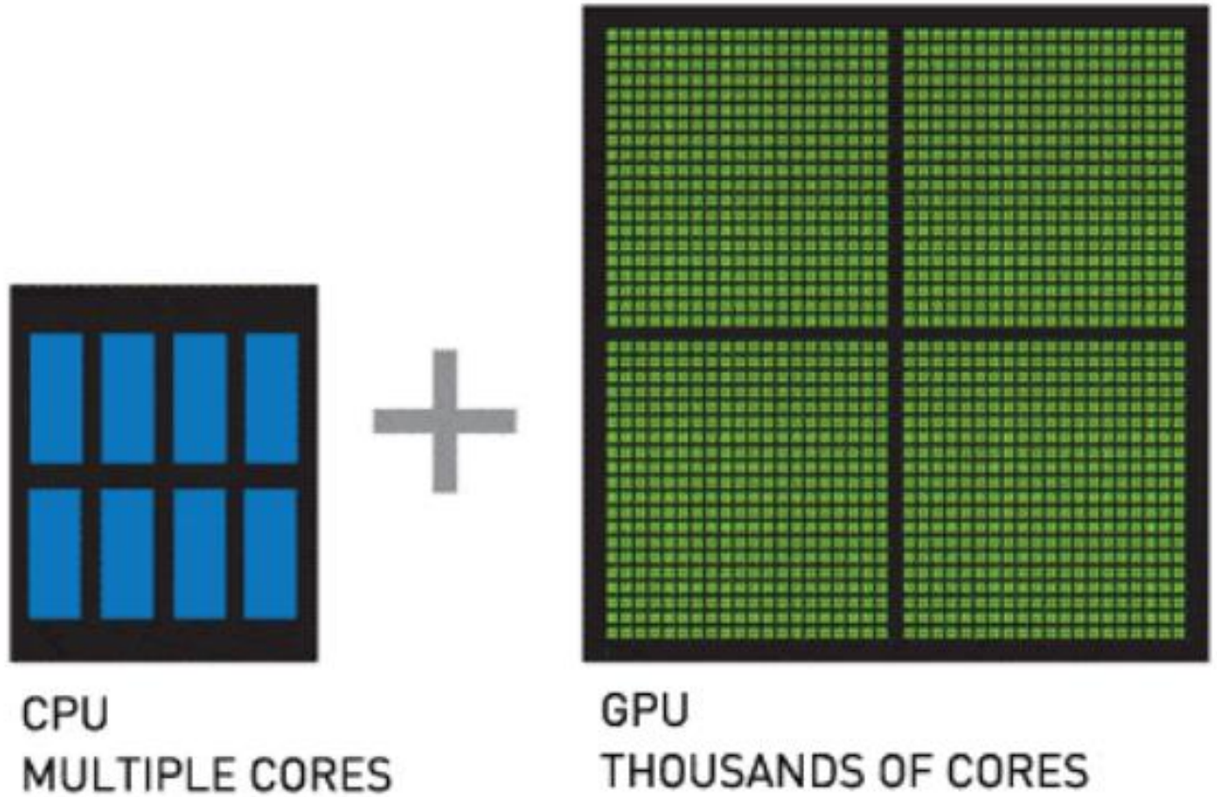
  - Loads/stores e operações são simplesmente tão amplos quanto a ALU mais ampla, portanto a ALU pode ser usada para calcular um valor de 128 bits, dois valores de 64 bits ou quatro valores de 32 bits

Classificação

Classificação de Arquitectura de Computadores

Taxonomia de Flynn

- Graphics Processing Units (GPU)
  - Qual é o processador com maior poder de computação no seu PC?

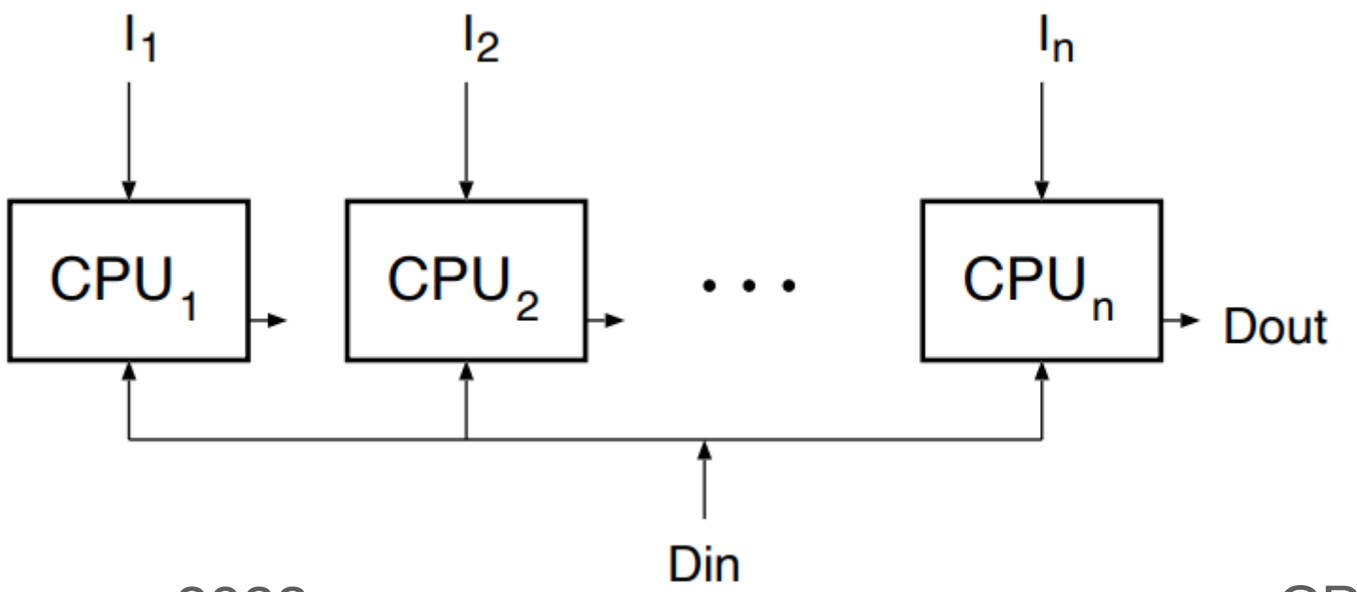


Classificação

Classificação de Arquitectura de Computadores

Taxonomia de Flynn

- Multiple Instruction, Single Data (MISD)
  - Cada elemento de processamento executa uma instrução diferente, mas todas as instruções operam nos mesmos dados simultaneamente
  - Não existem soluções comerciais deste tipo, as mais próximas desta classe são o processamento de pipelining e as matrizes sistólicas

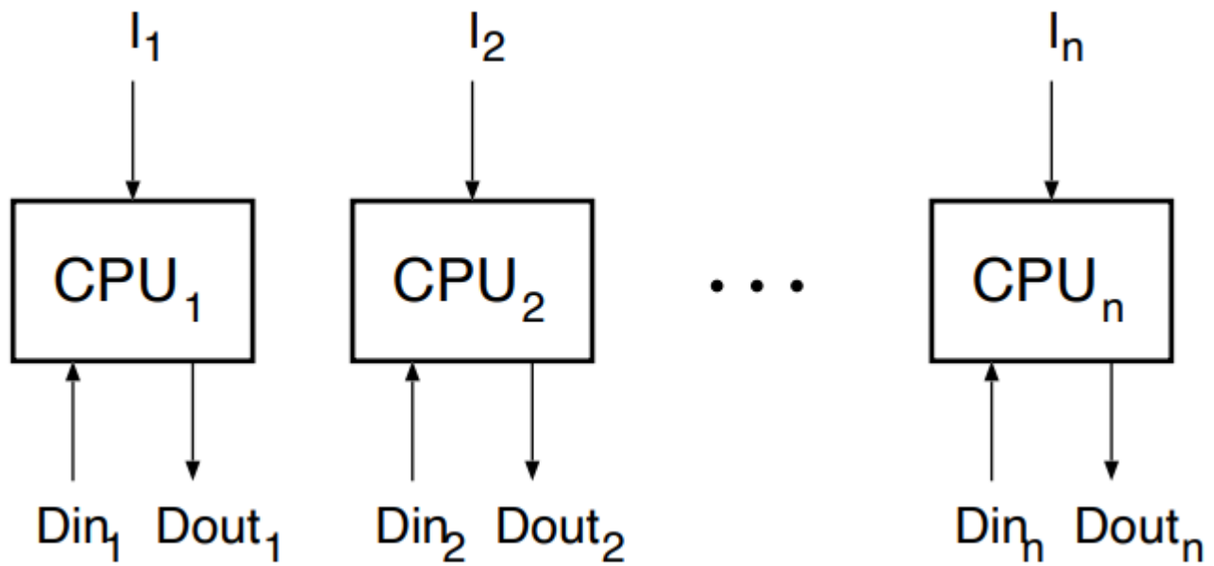


Classificação

Classificação de Arquitectura de Computadores

Taxonomia de Flynn

- Multiple Instruction, Multiple Data (MIMD)
  - Cada elemento de processamento executa instruções independentes em dados independentes (maior flexibilidade)
  - Duas classes:
    - memoria compartilhada
    - memória distribuída



Áreas de aplicação

- Porquê se preocupar com computação paralela?
  - Demanda contínua por maior poder computacional de muitos domínios diferentes!
- Duas classes principais de problemas em computação paralela:
  - **(1/2) Problemas do Grande Desafio (PGD)**
    - Problemas que não podem ser resolvidos em um período de tempo razoável com os computadores actuais.

Áreas de aplicação

- Porquê se preocupar com computação paralela?
  - Demanda contínua por maior poder computacional de muitos domínios diferentes!
- Duas classes principais de problemas em computação paralela:
  - **(2/2) Problemas Embaraçosamente Paralelos (PEP)**
    - Problemas cuja carga de trabalho pode ser facilmente dividida em tarefas (quase) independentes.

PGD

- Modelagem ambiental/ecossistêmica global
- Biomecânica e imagem biomédica
- dinâmica de fluidos
- nanotecnologia molecular
- Simulações de energia nuclear e armas

PEP

- Previsão numérica do tempo
- Computação gráfica / animação
- Ferramenta básica de busca de alinhamento local (BLAST) em bioinformática
- Métodos de Monte-Carlo
- Algorítmos genéticos

Exemplo (1/4)

Previsão de tempo

- A atmosfera é modelada dividindo-a em células tridimensionais (por exemplo, $1m^3$).
- O tempo é discretizado em intervalos (1 segundo, 1 minuto, 1 hora)
- As condições atmosféricas (temperatura, pressão, humidade, etc) para cada célula são calculadas em função das condições das células vizinhas neste e em intervalos de tempo anteriores.

Exemplo (2/4)

Previsão de tempo

- Para a previsão de Portugal continental, considere uma área de 1000km × 500km = 5×10^5 km^2.
- Assumindo uma altura atmosférica de 20 km, o volume total é de 10^7 km^3.
- Vamos considerar células cúbicas com 100 m de lado, portanto, temos um total de 10^{10} células para calcular.
- Se cada célula leva 25 operações de ponto flutuante, precisamos de um total de 25×10^{10} operações em cada intervalo de tempo.

Exemplo (3/4)

Previsão de tempo

- Se o segundo for o intervalo de tempo e quisermos calcular a previsão para amanhã (quase 10^5 segundos em um dia), um total de 25×10^{15} operações serão necessárias.
- Um Intel Core i7 de 3,5 GHz executa a 25 GFLOPS, portanto, esse cálculo leva cerca de 10^6s ou cerca de 10 dias...

Exemplo (4/4)

Problema n-corpo

- Cada corpo tem uma dada posição, velocidade, aceleração, que precisa ser computada para cada intervalo de tempo.

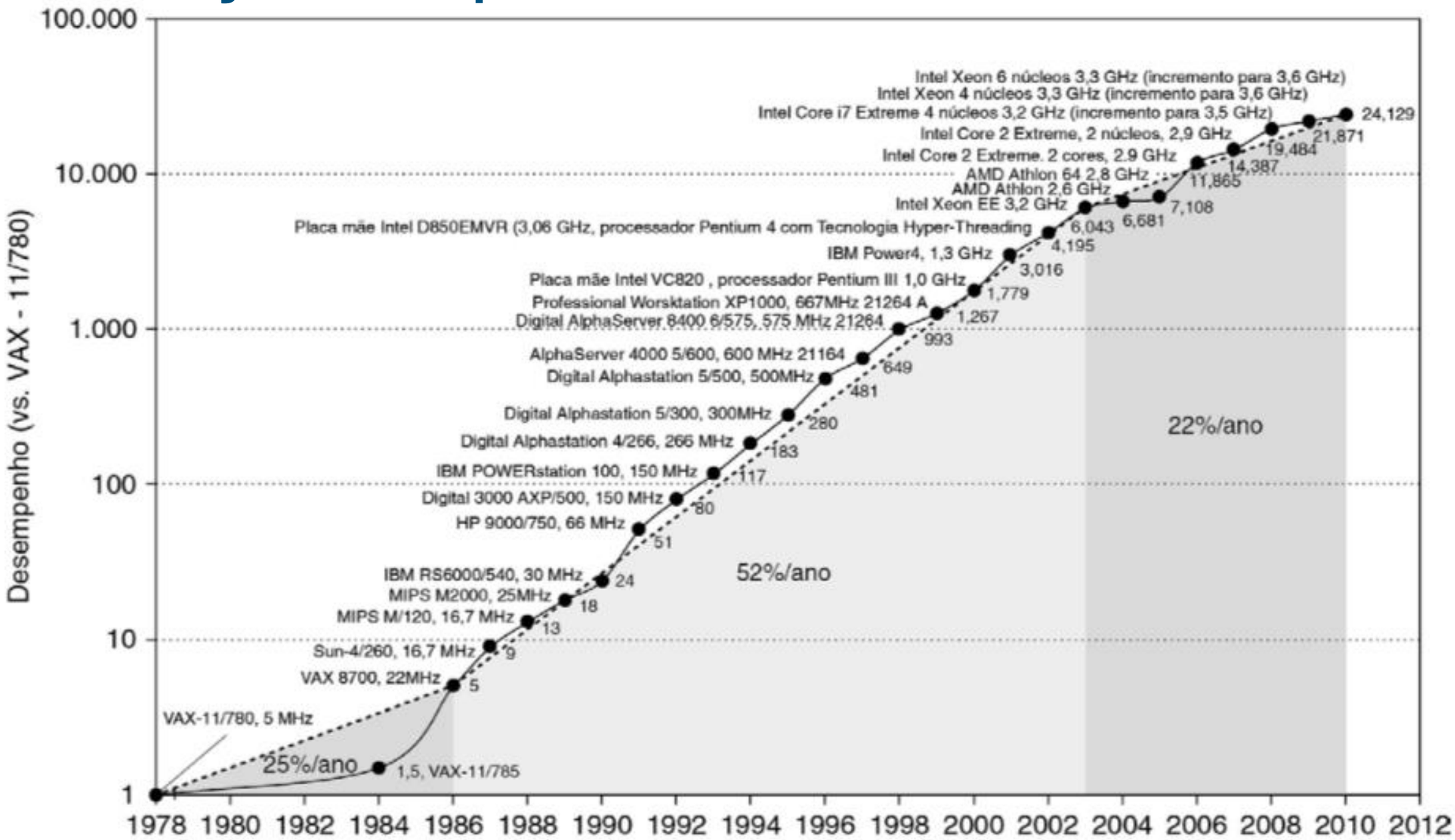
- Cada corpo atrai (e/ou repele) todos os outros corpos. Para n corpos, há um total de n^2 interações que precisam ser consideradas.

- Exemplo: uma galáxia possui mais de 10^{11} estrelas, levando a mais de 10^{22} operações de ponto flutuante para cada intervalo de tempo!

Evolução

(Em Portuguese do Brasil) David A. Patterson - Arquitetura de Computadores-CAMPUS - GRUPO ELSEVIER (2012), Capítulo 1, Secção 1.1.

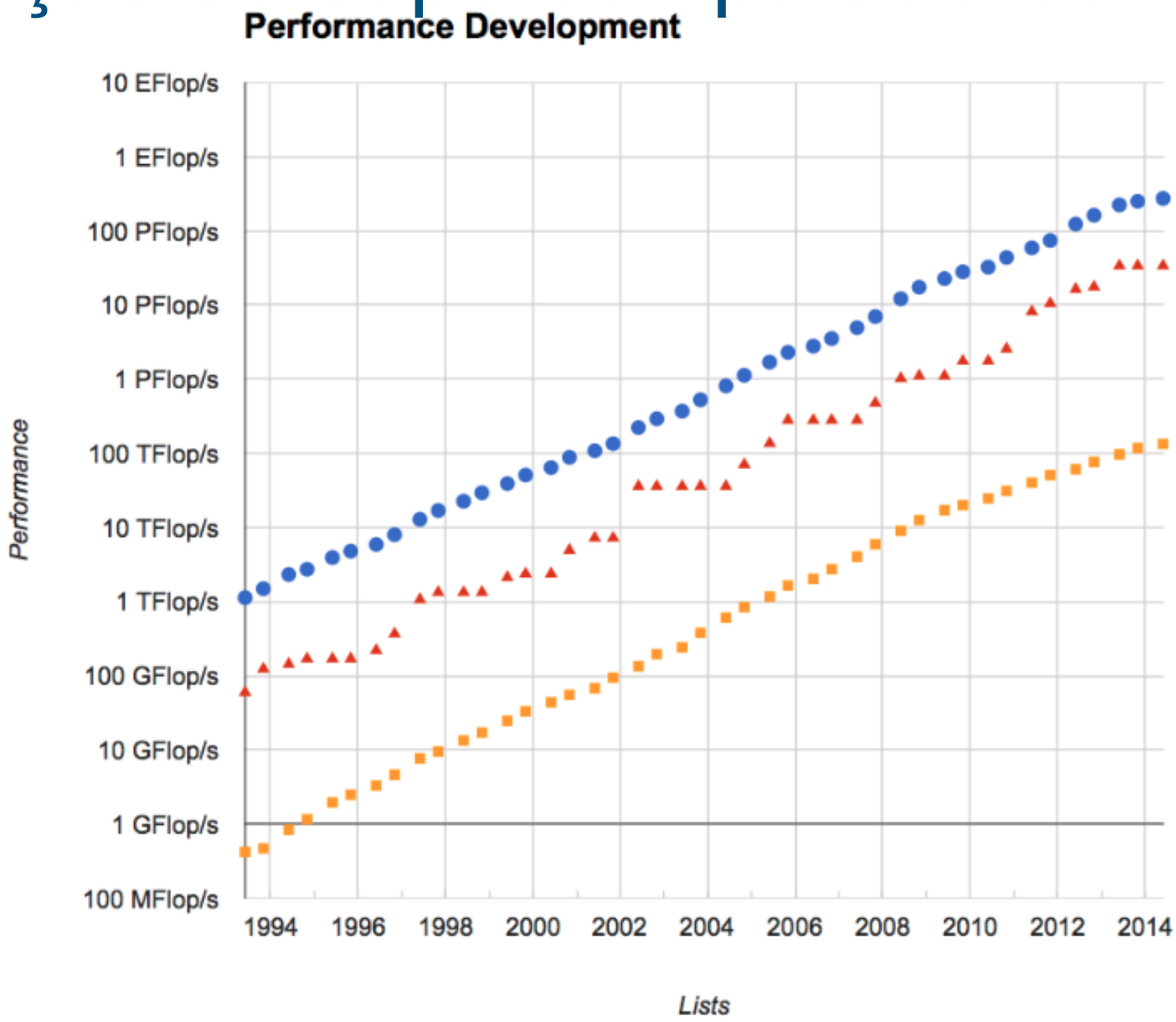
Evolução do processador



Evolução

https://top500.org/lists/top500/2014/11/highlights/

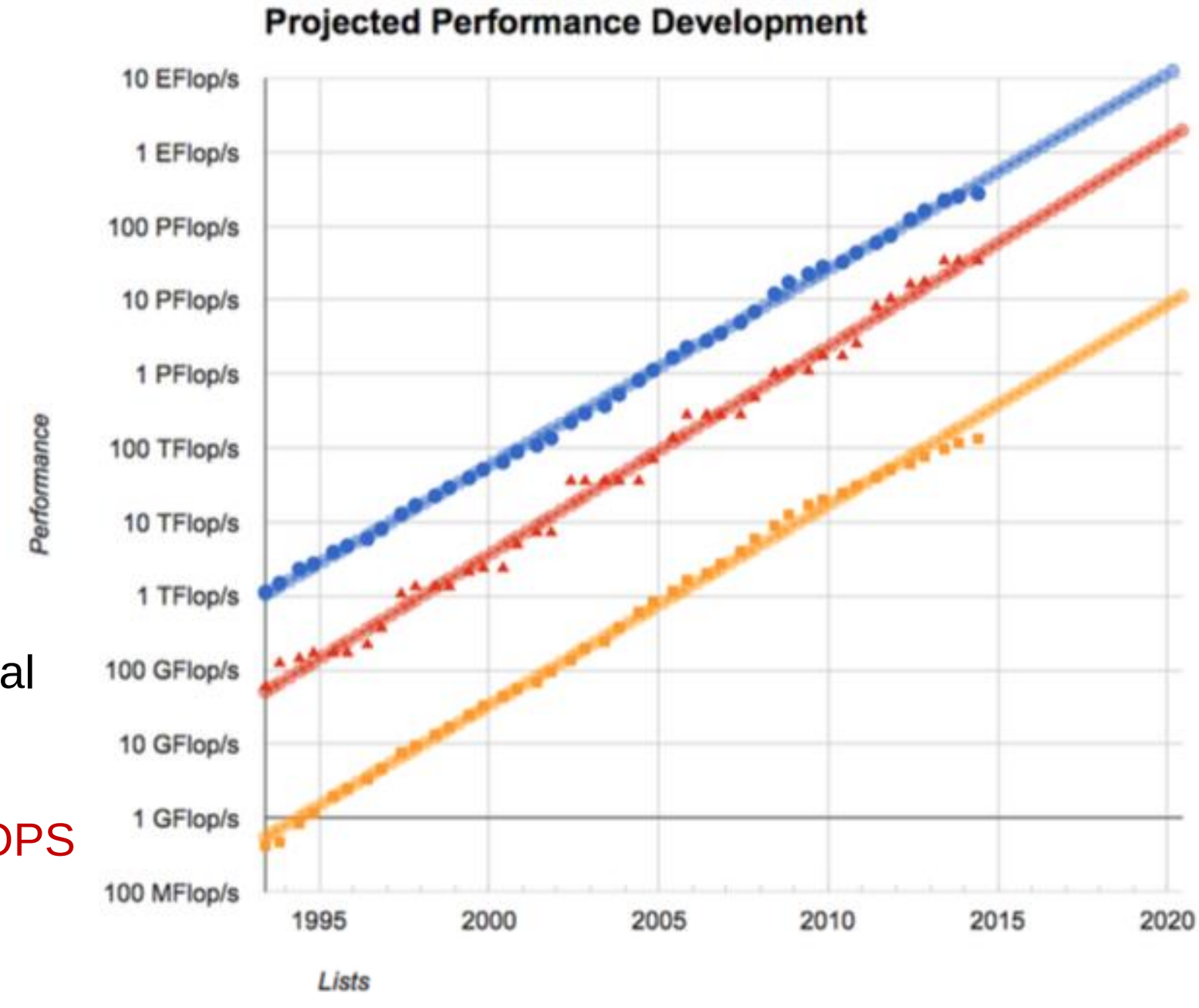
Evolução de supercomputadores



Evolução

Desenvolvimento de desempenho projectado

Primeiro sistema Peta disponível em 2009!



Estimativa do poder computacional do cérebro humano:
10^{14} conexões neurais a 200 cálculos por Segundo -> 20 PFLOPS

Supercomputadores

TOP500 LIST - JUNE 2022

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.

https://www.top500.org/lists/top500/list/2022/06/

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	8,730,112	1,102.00	1,685.65	21,100
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	1,110,144	151.90	214.35	2,942

Supercomputadores

Tipos de supercomputadores

- Processor Arrays (SIMD)
  - Nome associado ao processamento vetorial, muito popular nos primeiros supercomputadores.
- Multicore (SMP)
  - Conjunto de processadores que partilham uma memória principal comum.
- Massively Parallel Processors (MPP)
  - Processadores com memória principal individual com interconexões fortemente acopladas.

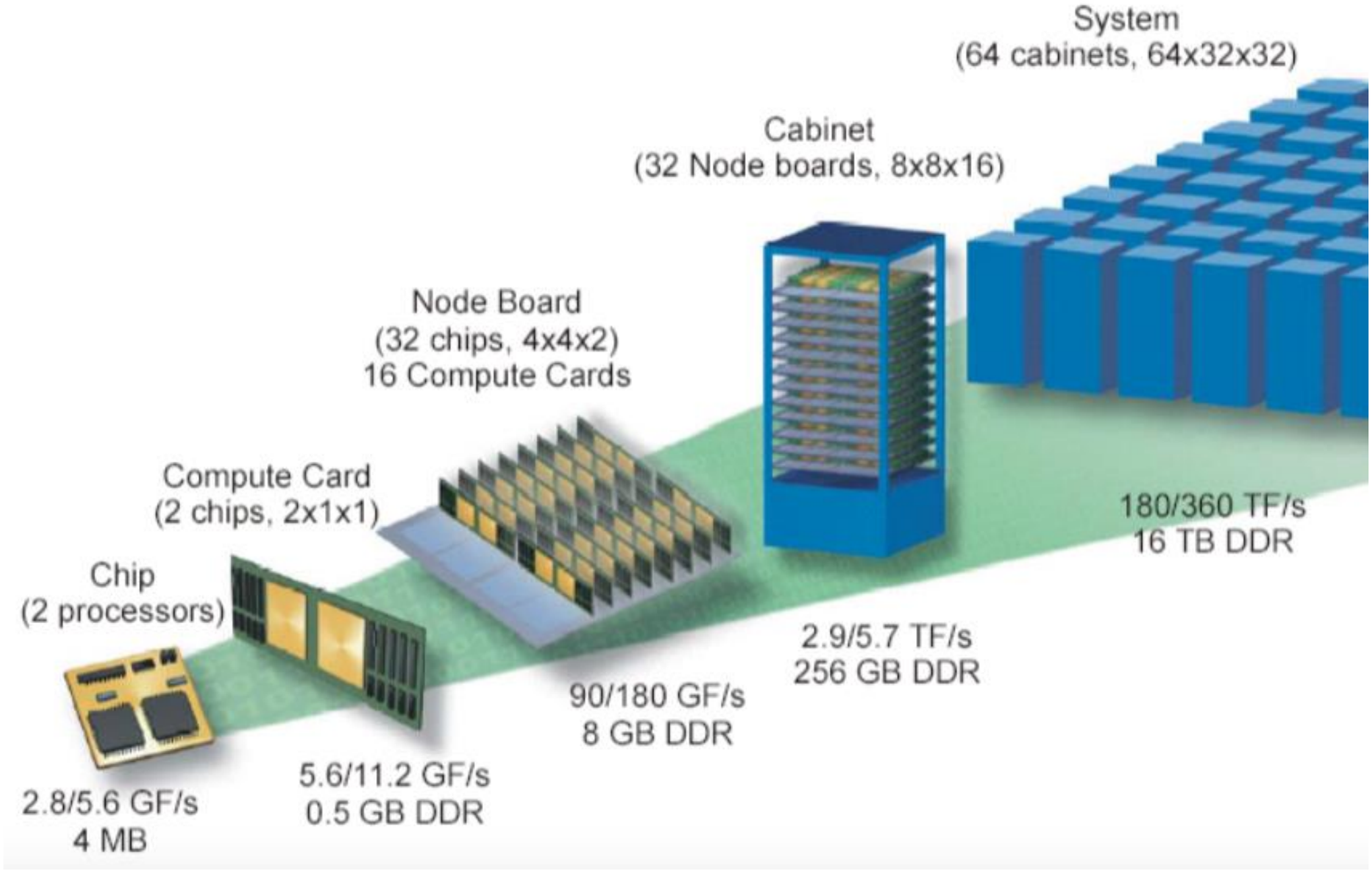
Supercomputadores

Tipos de supercomputadores

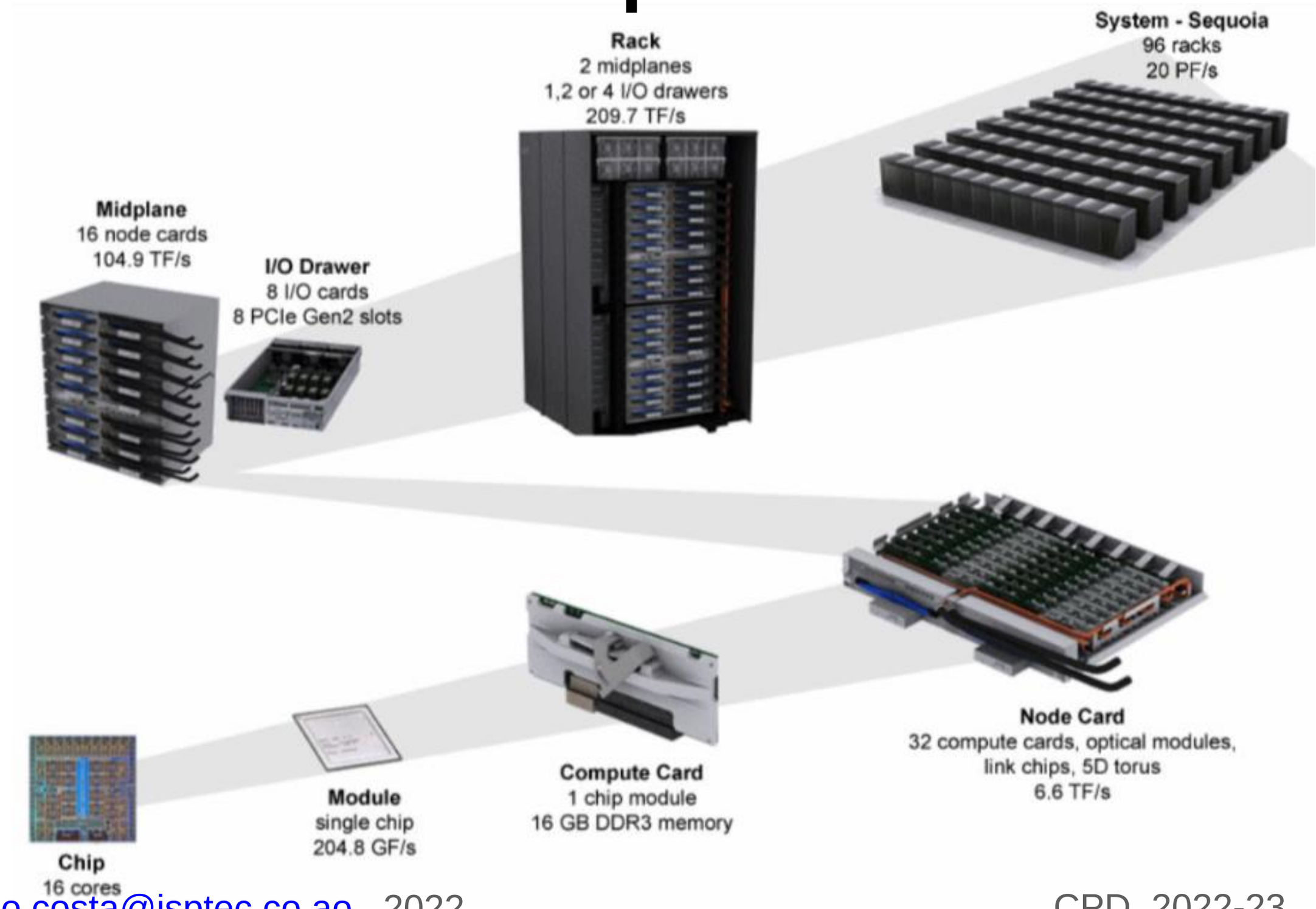
- Clusters
  - Processadores com memória principal individual interligados usando conexões InfiniBand, Quadrics, Myrinet ou Gigabit Ethernet.
    - COW / NOW: Cluster / Network Of Workstations
    - Beowulf: cluster made of PCs running Linux using TCP/IP (COTS: Commodity-Off-The-Shelf)
- Constelação
  - Cluster onde cada nó possui um grande número de núcleos.

Poder Computacional

Hierarquia do Poder Computacional

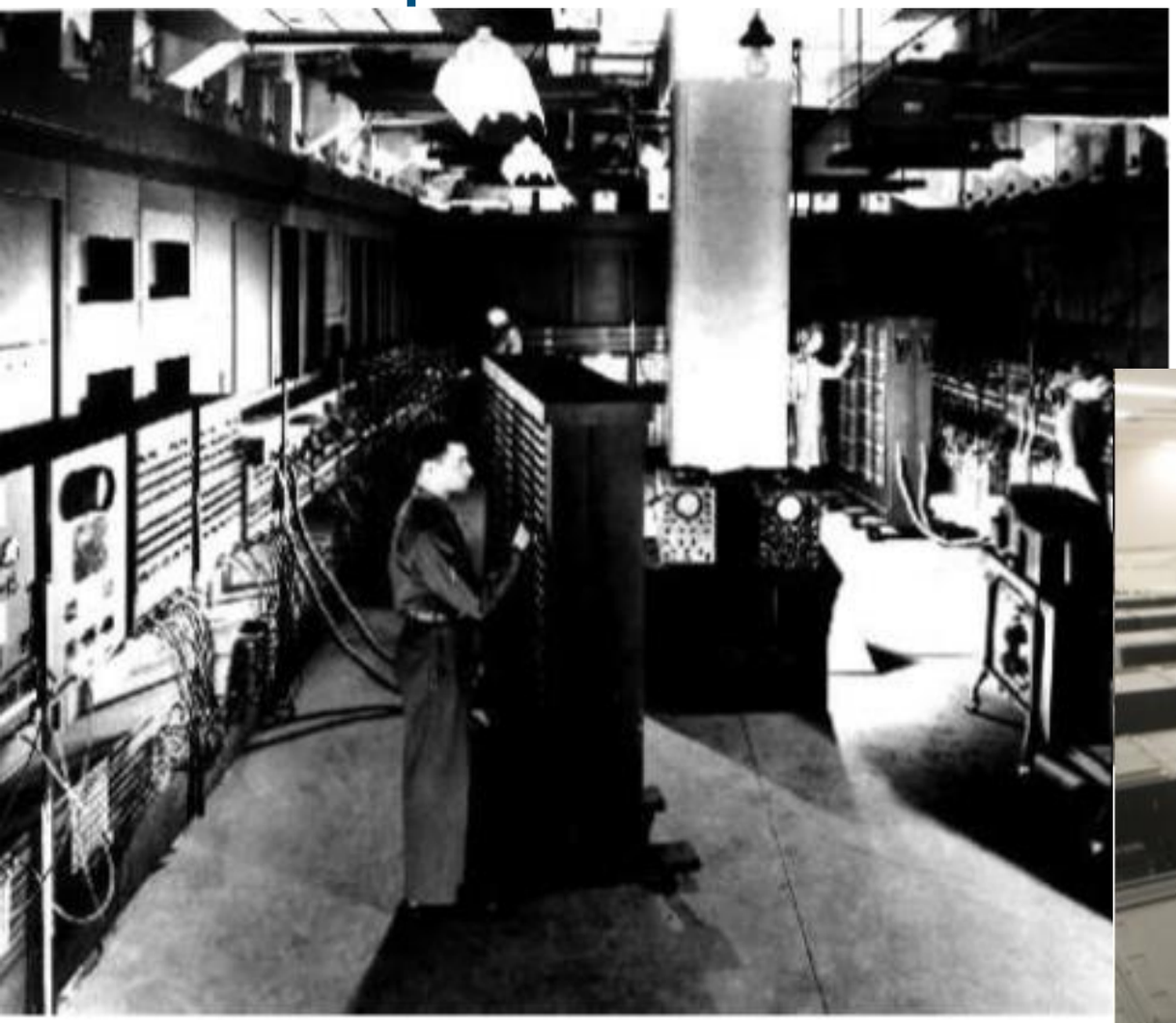


Poder Computacional



Poder Computacional

Computadores do tamanho de um armazém



Multicores

Amostra dos processadores multicore:

- AMD
  - Opteron: dual, quad, hex, 8-, 12- and 16-cores
  - Phenom: dual, quad, hex cores
- Intel
  - SandyBridge: up to 8 hyperthreaded cores
  - Haswell: up to 20 cores
- Oracle (ex-Sun)
  - Sparc T4/T5: 8/16 cores; 64/128 concurrent threads
- Oracle (ex-Sun)
  - Power 8: 12 cores
  - Cell: 1 PPC core; 8 SPEs w/ SIMD parallelism

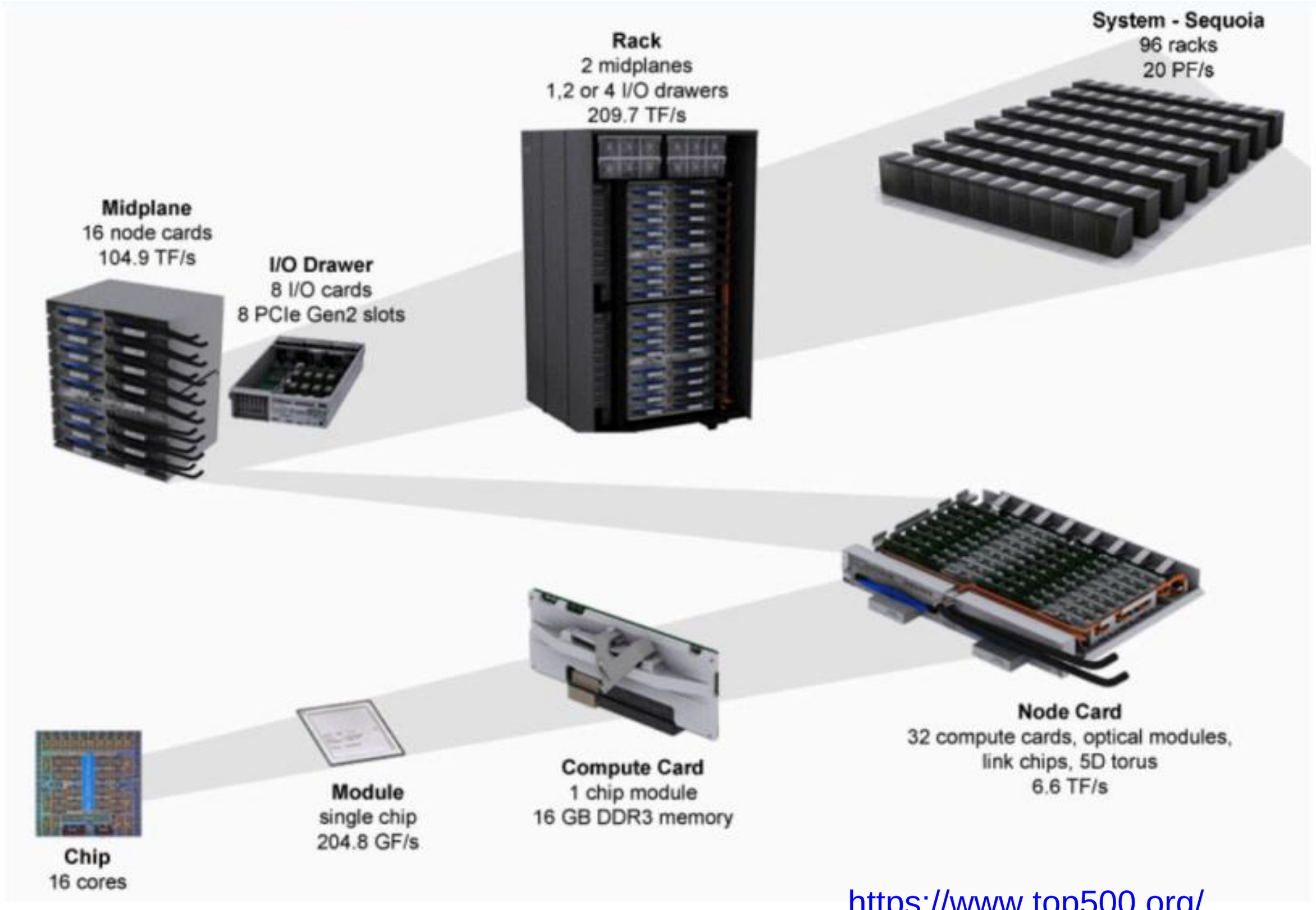
Níveis de paralelismo

- Instruções SIMD
- Execução paralela de diferentes estágios de instruções diferentes
  - Pipeline
- Processadores de múltiplos problemas (paralelismo de nível de instrução, ILP)
  - Palavra de instrução superescalar e muito grande (VLIW)

Níveis de paralelismo

- Vários processadores em um único chip, cada um a executar uma thread/processo independente
  - Processadores multicore
- Vários processadores em uma placa comum, cada um a executar uma thread/processo independente
  - Placas-mãe de multiprocessadores
- Execução paralela em computadores diferentes
  - Clusters, grades

Sequoia - Blue Gene/Q Supercomputer



https://www.top500.org/

Aceleradores

- GPU, Graphics Processing Unit
  - processador vetorial massivo
- TPU, Tensor Processing Unit
  - processador optimizado para as operações utilizadas em redes neurais (aprendizado de máquina)
- FPGA, Field Programmable Gate-Array
  - circuito de caminho de dados programável

Obrigado