Advanced Institute for Artificial Intelligence

Deep Learning HPC

Advanced Institute for Artificial Intelligence – AI2

https://advancedinstitute.ai

Deep Learning HPC

Agenda

- HPC
- GPU
- Tensorflow
- Conteiner

Introdução a HPC

O que é Computação de alto Desempenho ou HPC (High Performance Computing)?

- É uma área de pesquisa que lida com desafios relacionados com execução de aplicações com alto custo computacional
  - Normalmente, tais aplicações são utilizadas com muita frequência, e qualquer melhoria de desempenho provocam impacto
- Avanço no desenvolvimento de arquiteturas computacionais com maior poder computacional
  - Uma arquitetura computacional mais moderna provê recursos para melhorar o desempenho, porém a utilização de tais recursos nem sempre é trivial

Introdução a HPC

Otimização de desempenho de aplicações

- Caracterização das demandas computacionais das aplicações, por exemplo, demanda por espaço de armazenamento, processamento, memória RAM, etc
- Segmentação da aplicação em parte menores para que sejam executadas simultaneamente em diferentes recursos de uma arquitetura (paralelismo)
- Desenvolvimento de camadas de software otimizadas para diversas arquiteturas computacionais
  - Tais camadas abstraem a complexidade do trabalho de otimização e permitem uso eficiente da arquitetura

Introdução a HPC

Impacto no tempo de resposta no uso de aplicações com alto custo computacional

- Aplicações de Aprendizagem Profunda (Deep Learning) normalmente são computacionalmente intensivas
  - Busca por hiperparâmetros
  - Treino do modelo com novos conjuntos de dados
  - Prototipação de modelos
- Quanto mais rápido uma aplicação de aprendizagem profunda é executada, ainda que com um ganho não tão expressivo, apresenta impacto alto no trabalho dos especialistas desse domínio

Introdução a HPC

Arquitetura uniprocessador

- Novas gerações de processadores aumentaram o desempenho (Lei de Moore)
  - Até o desempenho do processador atingir seu limite
- Tendências para aumento de desempenho
  - Mais unidades de processamento em um único processador
  - Mais processadores na mesma máquina
- Para obter melhor desempenho, os desenvolvedores DEVEM explorar o paralelismo.

Introdução a HPC

- Um computador paralelo é um sistema de computador que usa vários elementos de processamento simultaneamente de maneira cooperativa para resolver um problema computacional
- O processamento paralelo inclui técnicas e tecnologias que permitem calcular em paralelo
  - Hardware, redes, sistemas operacionais, bibliotecas paralelas, linguagens, compiladores, algoritmos, ferramentas,...
- O paralelismo é natural
  - Problemas de computação diferem em nível / tipo de paralelismo

Modelo de desempenho em processadores atuais

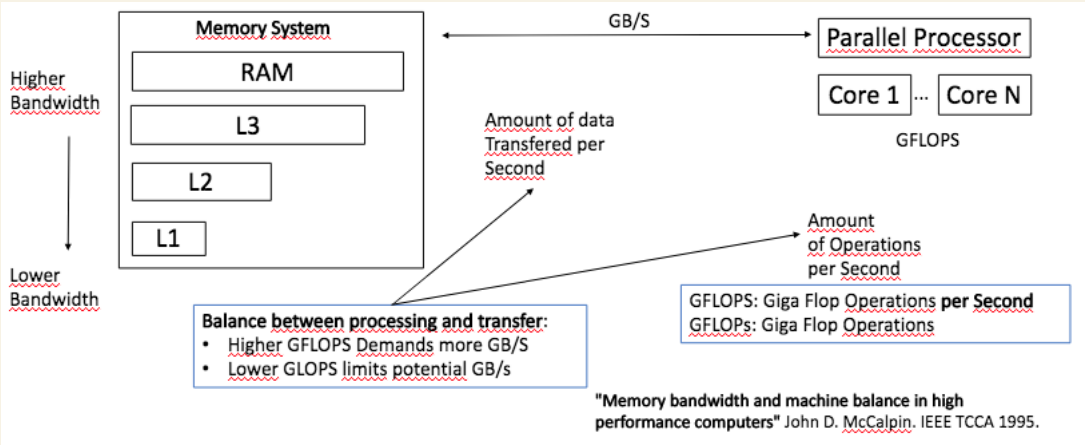
- Processador e sistema de memória trabalham de forma independente
- O desempenho da arquitetura computacional é medido de forma complementar
  - O desempenho do processador depende da eficiencia do sistema de memória entregar os dados para os registradores do processador
    - Se os dados são entregues de modo ineficiente, o processoador pode ficar ocioso em muitos momentos
  - O sistema de memória trabalha de modo eficiente se a vazão do processador é alta
    - Se o processoador não executa as operações de modo eficiente, o sistema de memória pode ficar ocioso em muits momentos

Introdução a HPC

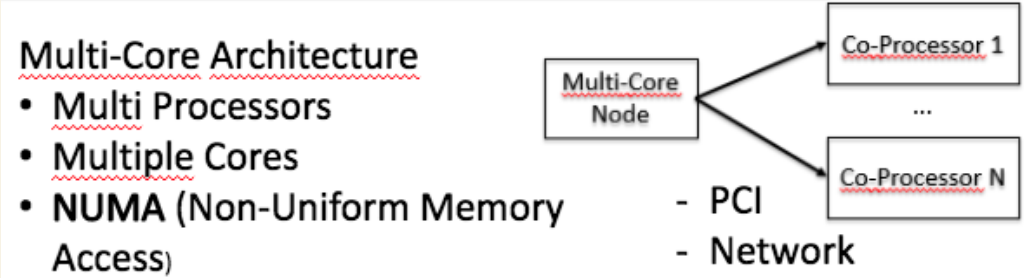
Técnicas para otimização de Desempenho

- Processador
  - Vetorização permite executar uma mesma intrução para conjuntos diferentes de dados
- Sistema de memória
  - Múltiplos níveis de memória permitem antecipar o envio de dados para o processador, aumentando a eficiência da execução das instruções

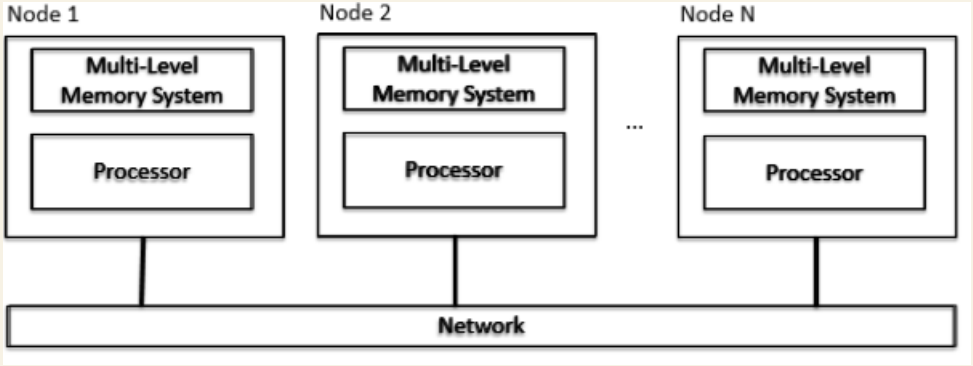
Introdução a HPC



As arquiteturas computacionais atuais, podem também ser compostas por coprocessadores ou aceleradores

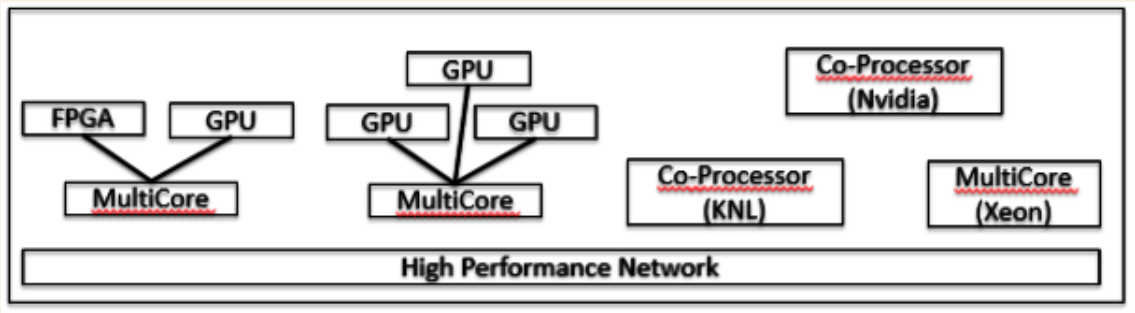


Uma forma de montar uma arquitetura computacional paralela é unificar diversos processador idênticos



Paralelismo

Outra forma de montar uma arquitetura computacional paralela é unificar diversos processador que podem ser diferentes entre si (Arquiteturas Heterogêneas)



Níveis de paralelismo

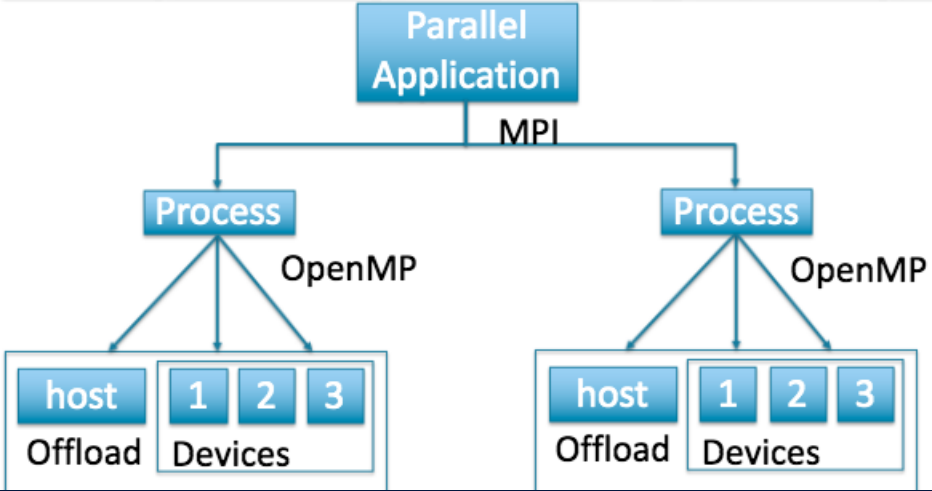
- Paralelismo no nível de instrução
  - Mecanismos do processador para aumentar o desempenho
- Paralelismo no nível de dados(Vetorização)
  - Execucao de mais de instrução identicas em paralelo para conjuntos distintos de dados, utilizando registradores vetoriais do processador
  - Vetorização em geral é combinada com técnicas de acesso eficiente utilizando múltiplos níveis de memória (cache)

Níveis de paralelismo

- Paralelismo no nível de tarefa (Thread) OpenMP
  - Múltiplas threads sendo executadas em paralelo
- Paralelismo no nível de processo
  - Mesmo programa em diferentes computadores coordenado a execução por troca de mensagens pela rede (MPI)
- offload
  - Dividindo o processamento entre o processador e um ou mais co-processadores, como GPU por exemplo

Paralelismo

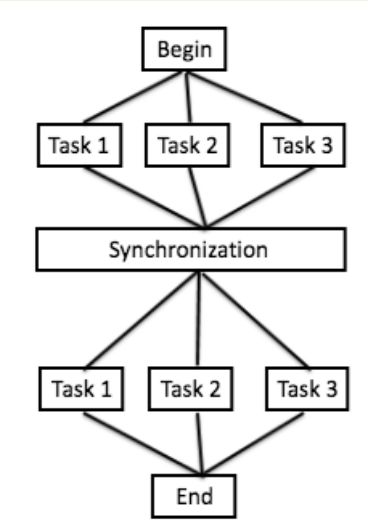
Os níveis de paralelismos podem ser usados de modo combinado por uma única aplicação



Paralelismo

- Para utilizar arquiteturas paralelas é necessário dividir a aplicação em tarefas independentes
  - As tarefas podem ser simultâneas e podem ser executadas ao mesmo tempo (execução paralela)
- As tarefas podem possuir algum nível de dependência, de tal forma que, uma tarefa pode necessitar de dados produzidos por outra tarefa antes de entrar em execução
  - Alguma forma de sincronização deve ser usada para impor (satisfazer) dependências
- No nível do software a paralelização deve considerar as sincronizações entre as tarefas
- Quanto ao uso do hardware é necessário traçar estratégias para utilizar de modo eficiente cada recurso de otimização de desempenho

Paralelismo

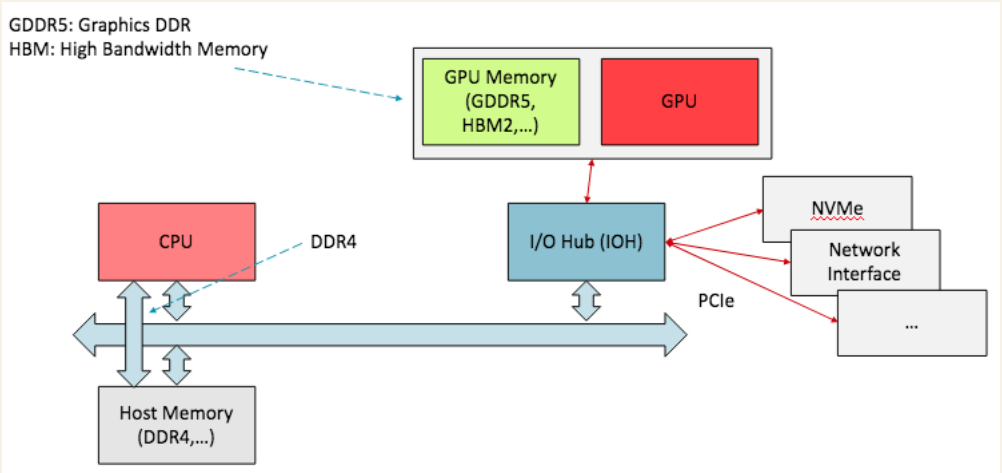


GPU

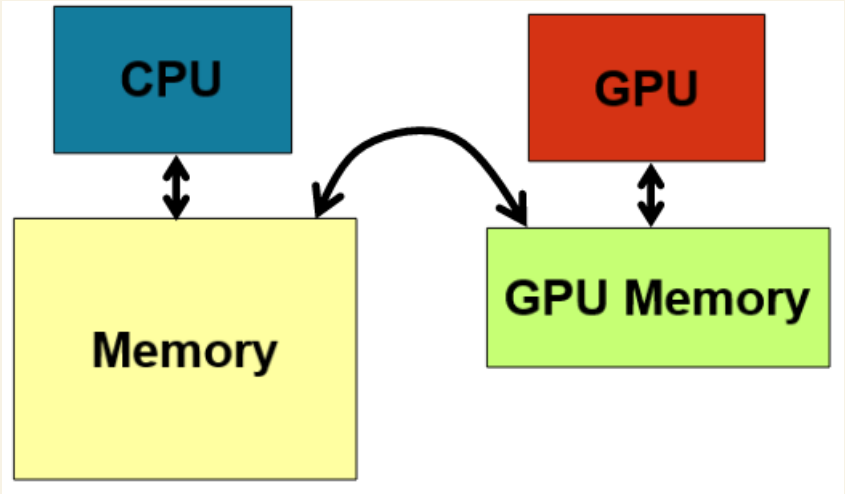
GPU

- É um multiprocessador paralelo / multithread otimizado para processamento de imagem.
  - O processamento gráfico é uma aplicação massivamente paralela
- GPGPU
  - Computação de uso geral usando GPU
- A GPU serve como um processador gráfico programável e como uma plataforma de computação paralela escalável.
  - Os sistemas podem combinar CPU + GPU para executar aplicativos

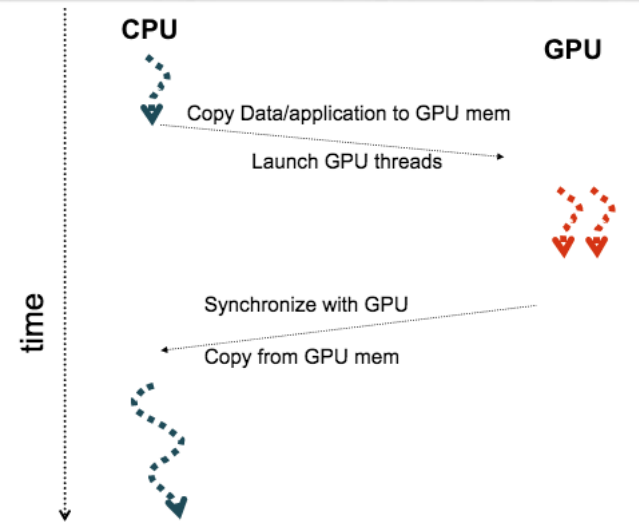
GPU


GDDR5: Graphics DDR
HBM: High Bandwidth Memory
GPU Memory (GDDR5, HBM2,...)
GPU
CPU
DDR4
I/O Hub (IOH)
NVMe
Network Interface
...
PCIe
Host Memory (DDR4,...)

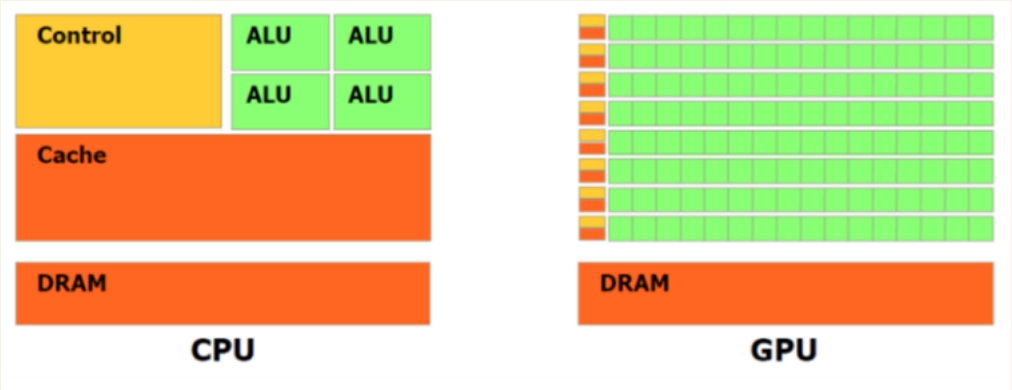
GPU



GPU


CPU
GPU
Copy Data/application to GPU mem
Launch GPU threads
Synchronize with GPU
Copy from GPU mem
time

GPU



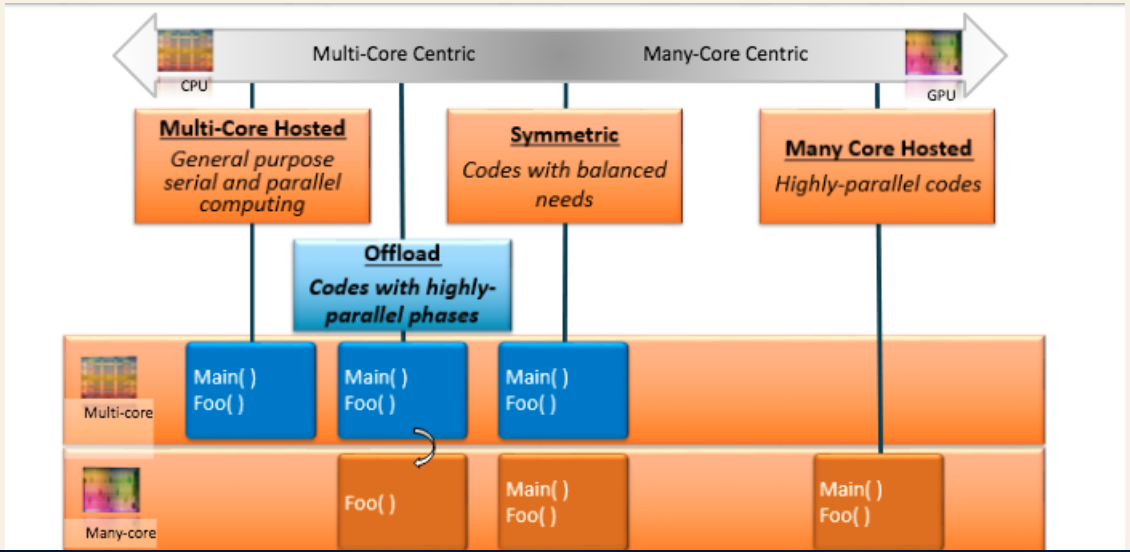
GPU

- lógica simplificada (sem execução fora de ordem, sem previsão de ramificação) significa que muito mais do chip é dedicado à computação de ponto flutuante
  - Núcleo da CPU Core x Núcleo da GPU
- Organizados como várias unidades, com cada unidade sendo efetivamente uma unidade vetorial, todos os núcleos fazendo a mesma coisa ao mesmo tempo
  - Kernel: uma rotina paralela para executar no hardware paralelo
- Maior largura de banda de memória que a CPU

GPU

- Objetivo não geral
  - Aplicações massivamente paralelas
    - Processamento gráfico
  - Aplicativos que exploram a localidade da memória
    - Cada unidade paralela realiza acesso ao seu próprio subconjunto de dados
    - Os algoritmos de paralelo de dados utilizam atributos da GPU
- Grandes matrizes de dados, taxa de transferência de streaming
- Cálculo de ponto flutuante de baixa latência (FP)

GPU


Multi-Core Centric
Many-Core Centric
CPU
GPU
Multi-Core Hosted
General purpose serial and parallel computing
Symmetric
Codes with balanced needs
Many Core Hosted
Highly-parallel codes
Offload
Codes with highly-parallel phases
Multi-core
Many-core
Main( )
Foo( )
Main( )
Foo( )
Main( )
Foo( )
Foo( )
Main( )
Foo( )
Main( )
Foo( )

GPU

- Como usar os recursos da GPU
  - Bibliotecas
    - Cublas
    - Tensorflow
  - Extensões da linguagem (diretivas)
    - OpenMP, OpenACC, OpenCL
  - fácil de otimizar código
  - Flexibilidade mínima
- Linguagem de Programação
  - API Cuda
  - Flexibilidade máxima
  - Acesso de baixo nível

GPU

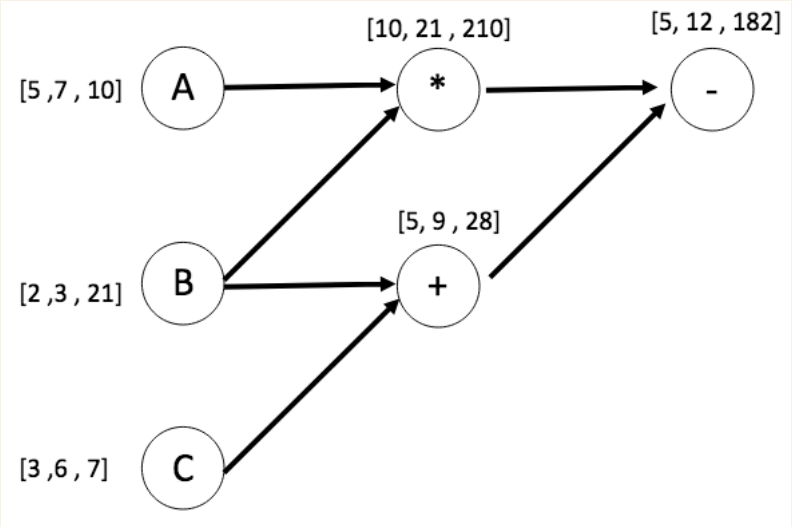
Bibliotecas para Deep Learning

- Tensorflow
- Keras
- Caffe2
- Rapids

Tensorflow

- Biblioteca de software de código aberto para computação numérica baseada no paradigma dataflow
- Execução de código otimizada porém acessível via API em um programa Python
- Portabilidade: distribui a computação em uma ou mais CPUs ou GPUs em um nó

Paradigma Dataflow



Montagem de Sistema

- Um grande desafio em colocar aplicações em produção é montar o ambiente adequado de Sistema opearacional, versões de bibliotecas e parametrizações do sistema.
- Quando há duas ou mais aplicações para colocar em produção o problema se torna ainda maior, pois um único ambiente deve atender duas demandas potencialmente distintas de sistema operacional
- Uma solução para esse problema é a virtualização, que refere-se a mecanismo de criar uma visão do sistema operacional para cada aplicação

Virtualização

- Um mecanismo de virtualização muito simples são as maquinas virtuais
- Outro mecanismo mais simples os ambientes virtuais como o conda
- Duas desvantagens desses dois métodos:
  - Máquinas virtuais prejudicam o desempenho das aplicações
  - Ambientes virtuais são desprovidos de flexibilidade quanto a configuração do sistema operacional

Linux Containers (LXC)

- Sistemas operacionais modernos oferecem recursos de virtualização no nível do sistema operacional
- Tal virtualização (chamadas conteineres) parte da premissa de que o kernel do SO permite a existência de múltiplas instâncias isoladas do espaço do usuário
- Tais instâncias permitem criar um ambiente de SO próprio que acessa os recursos do computador e do SO instalado na máquina
- Do ponto de vista dos programas em execução neles, parecem computadores reais
- Conteineres são mais rápidos de iniciar, produzem pouca sobrecarga no desempenho da aplicação e são muito flexíveis quanto a montagem do Sistema operacional

Docker

- Docker é uma ferramenta para facilitar o processo de criação de containeres
- A plataforma docker-hub [1] permite que usuários criem conteineres e compartilhem com outros usuários
  - Os comandos para gerenciar versões de conteineres docker é similar aos comandos do git hub
- Um conteiner pode ser compartilhado e novos containeres podem ser criadas a partir de um conteiner existente

[1] https://hub.docker.com/

Docker

Obter, rodar e terminar a execução de um conteiner

- Obter uma imagem (docker pull)
  - docker pull bulletinboard:1.0
  - Esse comando realiza o download da imagem do docker hub e controla a versão em uma pasta controlada pelo docker
- Rodar imagem (docker container run)
  - docker container run –name bb bulletinboard:1.0
- Terminar a execução de um conteiner
  - docker container rm –force bb

Docker

Criando novos conteineres docker

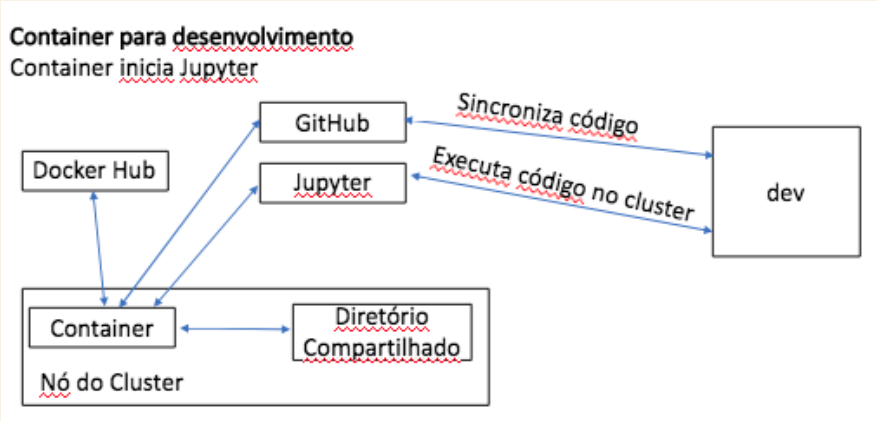
- Todo conteiner é definido por meio de um arquivo descritor chamado Dockerfile
- A partir de um descritor dockerfile é possível criar um conteiner
  - docker image build -t bulletinboard:1.0 .
- Para armazenar um novo conteiner que teve seu descritor alterado pode-se utilizar a opção tag
  - image tag bulletinboard:1.0 silviostanzani/bulletinboard:1.0
- Para efetivamente armazenar uma versão local do conteiner no docker hub deve-se utilizar push
  - docker image push silviostanzani/bulletinboard:1.0

Docker

Demonstração de uso de um container

- Criar uma imagem com streamlit e rede neural em Keras
- Armazenar no docker hub
- Recuperar e executar a imagem

Possível cenário de uso de Conteineres



Possível cenário de uso de Conteineres