Insper

SuperComputação

Aula 11 – Introdução a paralelismo

2021 – Engenharia

Igor Montagner <igorsm1@insper.edu.br>
Antônio Selvatici <antoniohps1@insper.edu.br>

Resolução de problemas

- Heurísticas
- Busca local
- Busca exaustiva
  - Branch and Bound (propriedades do problema)

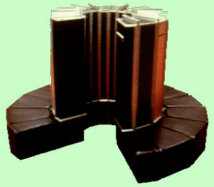
Solução de alto desempenho

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

Mas e o paralelismo?

Vector Computers

SMP computers

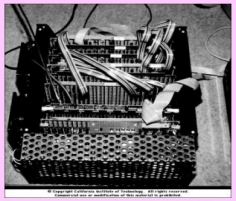


Cray 1 (1976)

Cray 2 (1985)

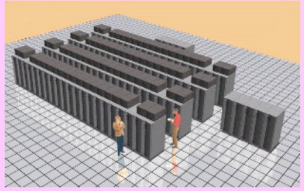
Cray C-90 (1991)

Massively Parallel Processors (MPP)



Cosmic cube (1983)

Paragon (1993)



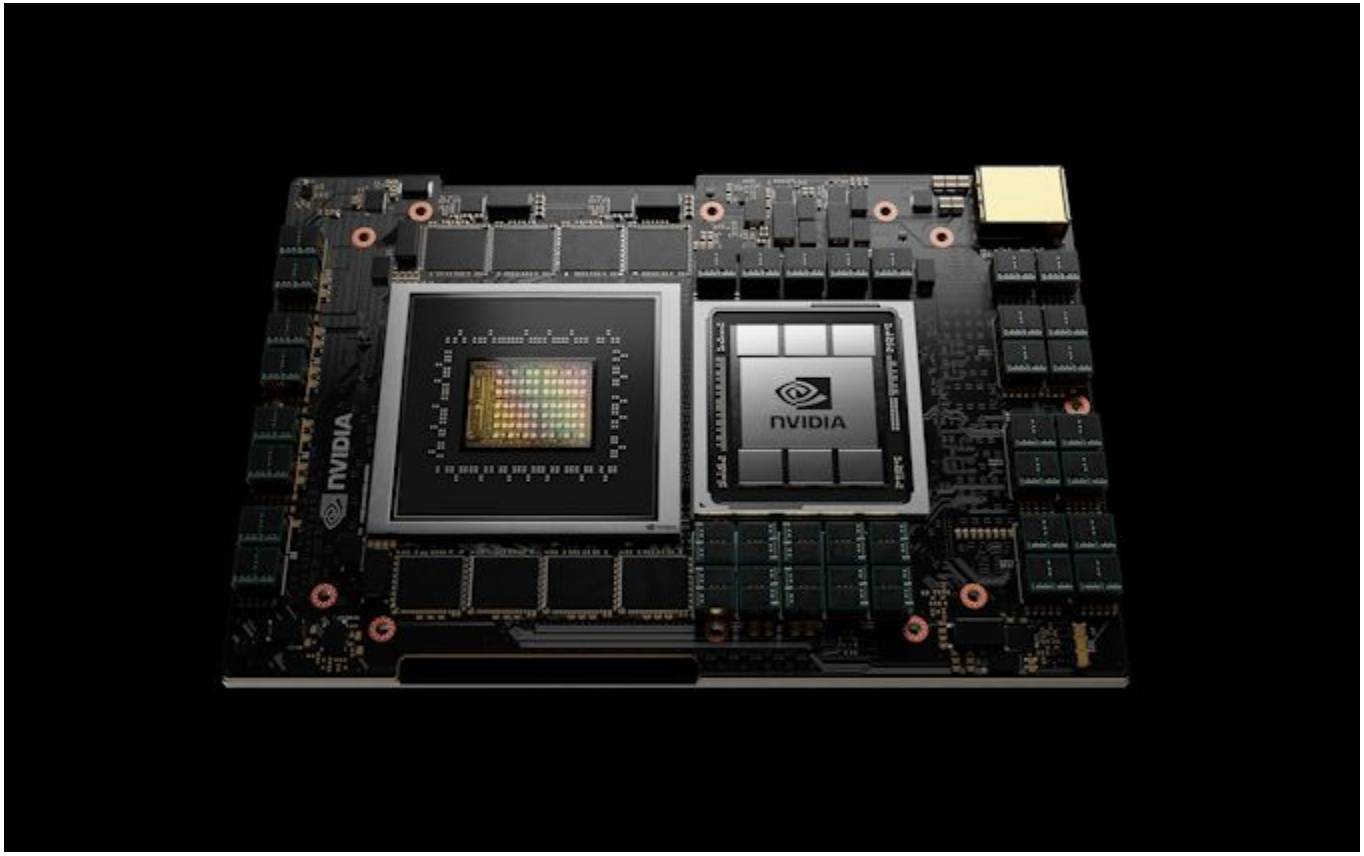
ASCI Red (1997)

Cluster Computers

Clusters (late 80's)

Linux PC Clusters
(~1995)

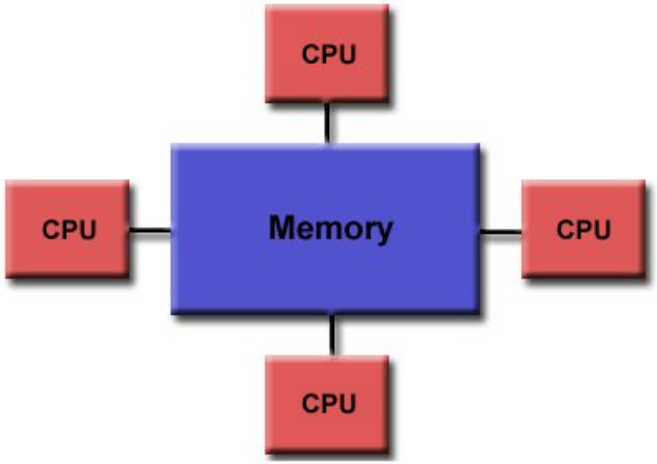
Novidades a frente



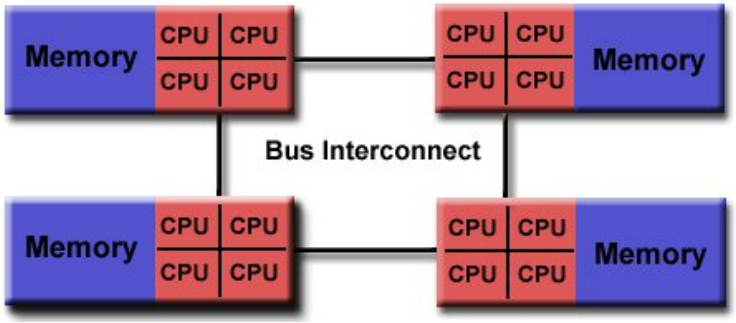
Nvidia Grace integra CPU-GPU em SoC rápido

Insper

Sistemas Multi-core



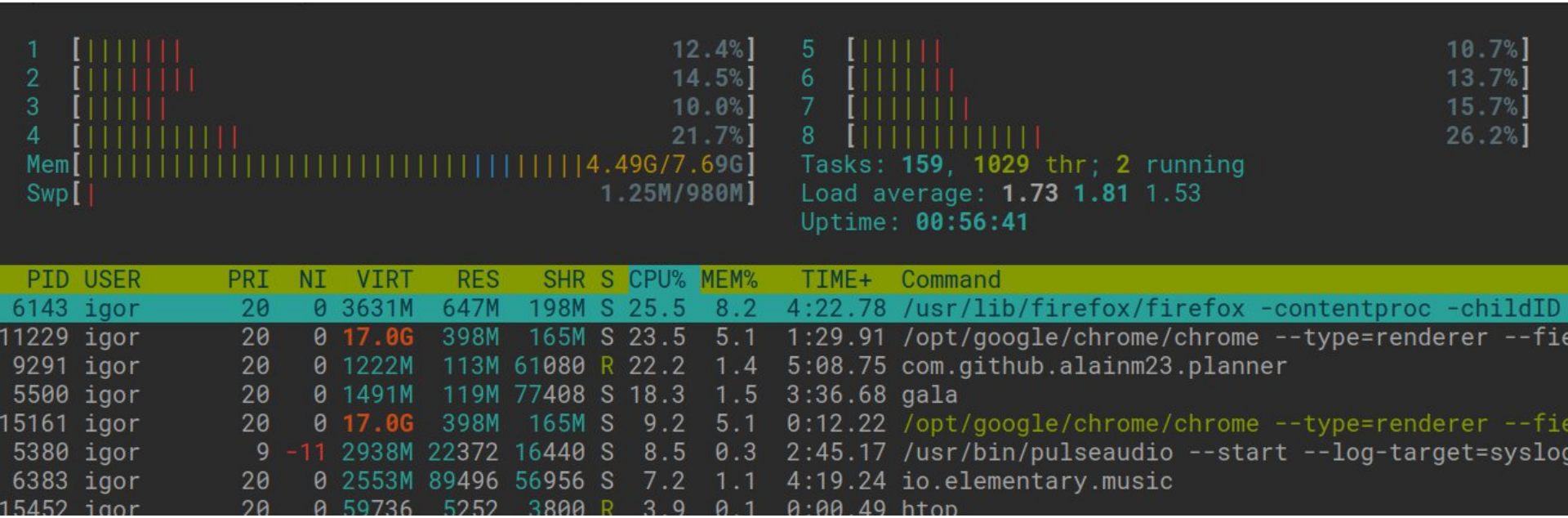
Uniform Memory Access



Non-Uniform Memory Access

Insper

Sistemas multi-core



Insper

Discussão I: qual expectativa de melhoria de velocidade?

Exemplo 1

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

Exemplo 1

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

Tempo total divido por 8!

Exemplo 2

```
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]);
}
```

Exemplo 2

```
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 :(

Conceito 1: Dependência

Um loop tem uma **dependência** de dados sua execução correta depende da ordem de sua execução.

Isto ocorre quando **uma iteração depende de resultados calculados em iterações** anteriores.

Quando não existe nenhuma dependência em um loop ele é dito **ingenuamente paralelizável**.

Exemplo 3

```
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]);
}
```

Exemplo 3

```
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 fazer resultados1 e resultados2 em paralelo!

Conceito 2: Paralelismo

Paralelismo de dados: faço em paralelo a mesma operação (lenta) para todos os elementos em um conjunto de dados (grande).

Paralelismo de tarefas: faço em paralelo duas (ou mais) tarefas independentes. Se houver dependências quebro em partes independentes e rodo em ordem.

Exemplo 4

```
std::vector<double> dados;

le_dados_do_disco(dados); // demora 10 segundos
// dados.size() == 100

for (int i = 0; i < dados.size(); i++) {
    operacao_complexa3(dados[i]); // demora 0,1 segundo
}
```

Quanto tempo o programa demora?

Existem relações de dependência?

Qual a expectativa de tempo para um programa paralelo?

Conceito 3: Lei de Amdahl

Dada uma tarefa que dura X horas, sendo que Y horas correspondem a trabalho que pode ser paralelizado, o número máximo de vezes que podemos acelerá-la é

$$\frac{1}{(1 - p)}$$

onde $p = \frac{Y}{X}$

Exemplo 4

```
std::vector<double> dados;

le_dados_do_disco(dados); // demora 10 segundos
// dados.size() == 100

for (int i = 0; i < dados.size(); i++) {
    operacao_complexa3(dados[i]); // demora 0,1 segundo
}
```

Quanto tempo o programa demora?

Existem relações de dependência?

Qual a expectativa de tempo para um programa paralelo?

Resumo

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

OpenMP

Paralelismo Multi-core

Threads:

- Compartilham memória
- Sincronização de acessos

Processos:

- Troca de mensagens
- Possível distribuir em vários nós

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 mínimo de modificações no código sequencial

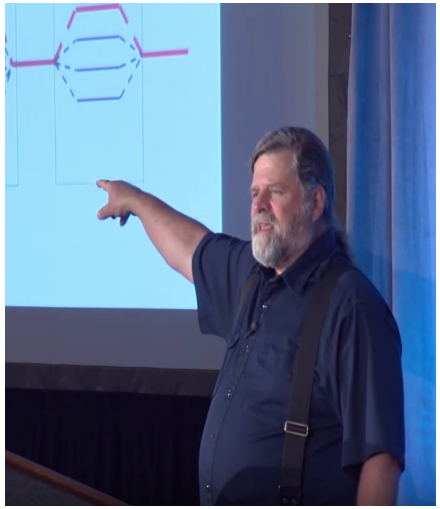
Fontes importantes

A brief Introduction to parallel programming

Tim Mattson

Intel Corp.

timothy.g.mattson@ intel.com



Vídeos:

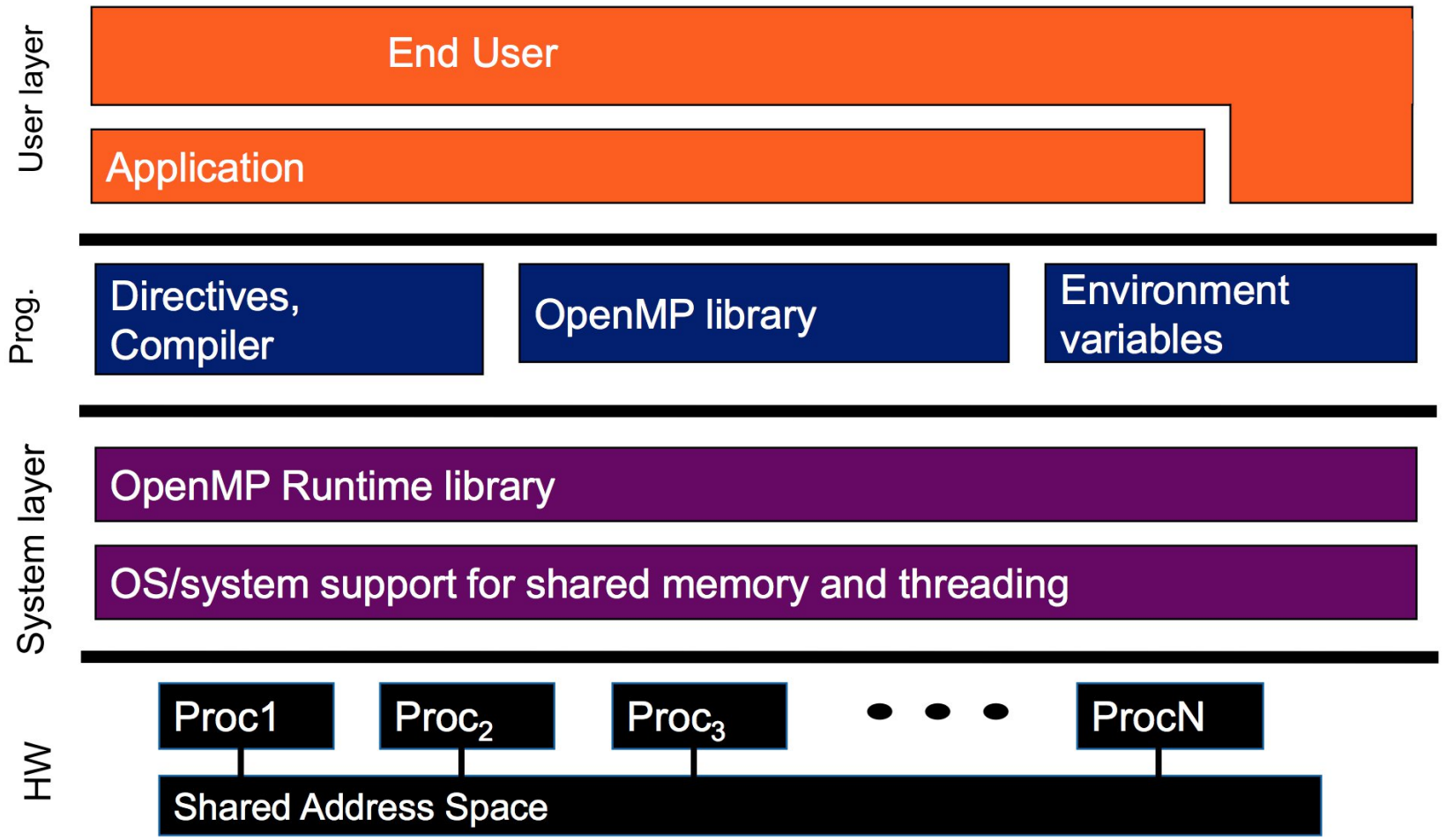
https://www.youtube.com/watch?v=pRtTIW9-Nr0
https://www.youtube.com/watch?v=LRsQHDAqPHA
https://www.youtube.com/watch?v=dK4PITrQtjY
https://www.youtube.com/watch?v=WvoMpG_QvBU

Slides:

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

Insper

OpenMP (host / NUMA)



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

Insper

www.insper.edu.br