Programação Paralela Avançada - PPA

* um dos quesitos da avaliação é ver se não tem erros no VALGRIND

Maurício Pillon

~~~ 06/08/2019 ~~~

Transparência: 01_Apresentacao_disciplina.pdf

Requisitos-de-avaliacao.pdf

PROVA (40%)	Projeto (30%)	Atividades Complementares (30%)
~~26/09/2019~~ 01/10/2019	27/08/2019	07/11/2019

Bibliografia Básica: Parallel Programming techniques and applications using network workstation and parallel computers } Barry Wilkinson

~~~ 08/08/2019 ~~~

Transparência: 02_Programação-RevisaoC.pdf

int = 4
float = 8
void = 1
} tamanho ponteiros

Um programa se torna um processo quando é alocado na memória

- Stack (valor estático)
- MMR (dinâmico malloc)
- Heap
- BSS
- Data
- Text

(man gcc) ↔ gcc -Wall -g -O3 10_revisaoC.c -o a.exe

- gcc → compilador e linkador: gera um compilado e associa o executável com as bibliotecas
- -Wall → warnings all
- -g → aviso
- -O3 → otimizador

antigamente os SO não liberavam as memórias usadas pelos programas, era necessário dar free.

.a → dinâmico
.o → estático

ponteiro:
- & : é unário e retorna o endereço da variável
- * : é unário e retorna o valor do conteúdo

Debugers

Valgrind ; GDB

ajuda a identificar problemas de memória.

identifica: Overflow

13/08/2019

Transparência: 02-Programacao-MatrizC.pdf

Soma; Subtração (0 de dependência) Soma e multiplicação de matrizes tem dependências zero, ou seja é fácil de paralelizar
Multiplicação; Divisão (um pouco mais de dependência)
dependência na soma

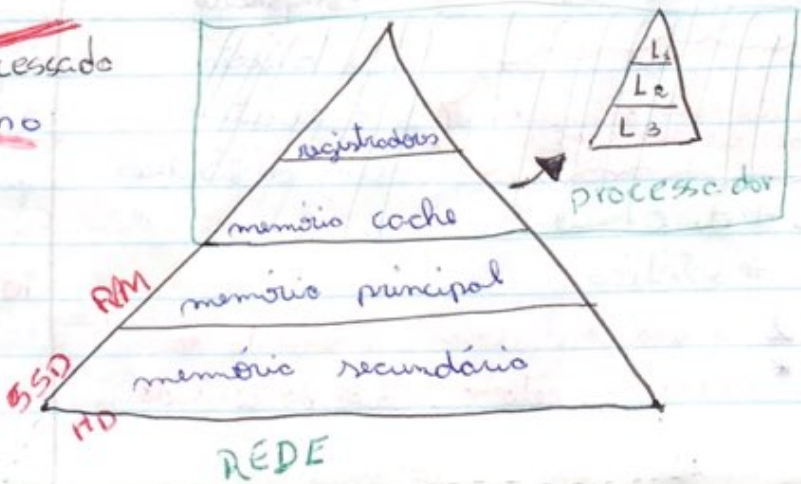
Mas devemos verificar as condições internas nos códigos

Multiplicação em Blocos
* da para cortar duas vezes (ou mais)
realizar cortes nas matriz, mas mantendo as condições para a multiplicação. * O mais difícil é definir o melhor corte

Transparência: 03_Tipos-de-Computadores-Paralelos.pdf

* Arquiteturas paralelas
  - Multiprocessadores (fortemente acoplado) com memória compartilhada
    - homogêneas
    - mais conveniente
  - Multicomputadores com memória distribuída (fracamente acoplado) (pode ser homogênea ou heterogênea)
    - cluster

~~Maquina Monoprocessada~~
~~Pseudo paralelismo~~



13/08/2019

* Trocar a ordem dos laços do for (i, j, K por K, j, i) pode ter um aumento de desempenho de até x8.

* algoritmo de coerência de cache (cache right back)
↳ antes de escrever no cache,
~~todos~~ avisa todos os vizinhos que o valor não é mais o mesmo

Instruções e Dados } algumas tem arquiteturas que tem caches exclusivos para dados e instruções

(slide 9) fim

15/08/2019

(slide 10) início

Transparência: 03-Tipos-de-Computadores-Paralelos.pdf

Troca de mensagem
Memória Compartilhada
Arquivos

Três formas de comunicação entre processos no mesmo PC

na *REDE só existe a troca de mensagens (slide 11)

(sempre overhead)
* Largura de banda
* Latência de rede
* Latência de comunicação
* Custo

Inviável (economicamente / tecnicamente)

Número de conexões Todos para Todos
C(c-1)/2 (slide 13)

* Cada máquina precisaria ter muitas placas de redes

Quais as formas viáveis?

Buffer = fila

15/08/2019

Diâmetro: número mínimo de ligações entre os dois nós mais distantes na rede

Soluções viáveis:

*MESH: quatro placas de rede por PC

$2(\sqrt{P}-1)$

diâmetro

exemplo:
$2(\sqrt{16}-1)$
número de pulos (slide 141)

*Hypercube: três placas de rede por PC
↳ quanto maior a dimensão do cubo, mais placas de rede

$\log_2 P$

diâmetro

Cross-bar: Conexão por Switch
Tree Network

dar de cabeça: balancear a rede e deixar os computadores coordenadores/próximos para não serem usados na rede

Multicomputador: Homogêneo vs Heterogêneo

Controle dos dados é responsabilidade ~~da aplicação~~ minha (programador)

hyperthred → não é boa só para CPU-BOUND (é boa para atenção)

20/08/2019

Transparência: 03-Tipos-de-Paralelismo-e-Particionamento.pdf
04-Classificacao-de-arquiteturas-paralelas.pdf

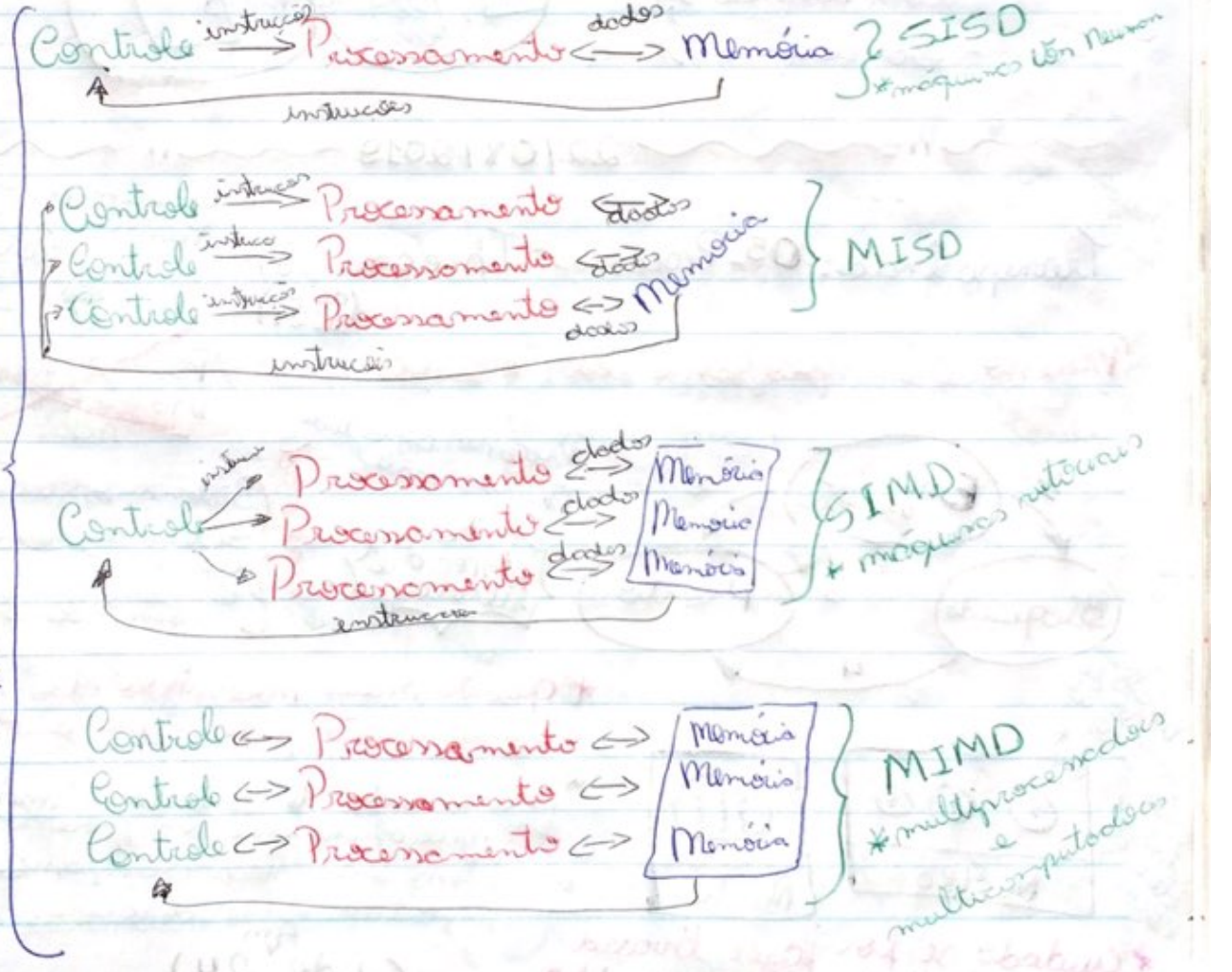
Classificação de Flynn (instruções e dados)

- SISD → monoprocessado
- MIMD → multiprocessado

	mono Dado	multi Dado
mono instrução	SISD	SIMD
multi instrução	MISD	MIMD

não tem sentido

Máquina Vetorial
- ~~CPU~~
- Placa de Vídeo (GPU)



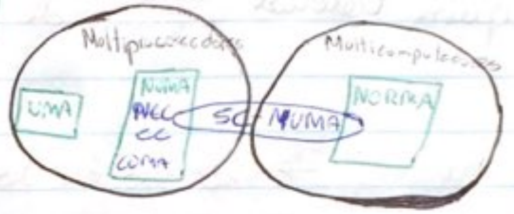
~~ 20/08/2019 ~~

No mesmo computador a memória sempre é dada pela menor velocidade dos RAMs **da** rede

em computadores diferente na rede isso não acontece!!!

Classificações de Memória: **UMA** ($t_1 = t_2$) * problemas sincronização e coerência
NUMA ($t_1 != t_2$) * Divisão baseada em coerência
NORMA ($t_1 != t_2$)

programador tem a responsabilidade de gerenciar *
- Sem coerência
- Com coerência H.
- Com coerência S.
- Só cache



~~ 22/08/2019 ~~

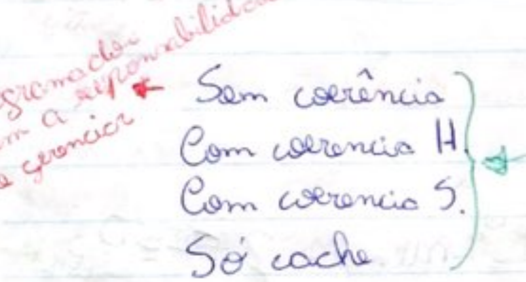
Threads são processos leves

Transparência: 05_Processos-Threads.pdf (do 0 até 24)

Queremos o Paralelismo real

~~PSEUDOPARALELISMO NÃO É INTERESSANTE AQUI~~

Processo é um programa em execução



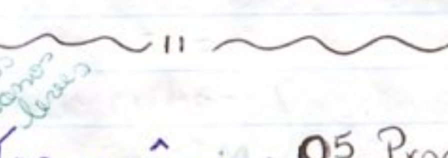
Processos copia tudo
Threads não

pthread é híbrida

escalonador
- Sistema operacional
- Sistema de gerenciamento de threads
- Eu posso gerenciar os meus processos

* quanto mais necessidade de comunicação, pior é para a thread

são alguns elementos do processo que estão em execução



* chaveamento de contexto é menor em threads do que em processos
* existem bibliotecas que permitem o compartilhamento de variáveis locais

fim (slide 24)

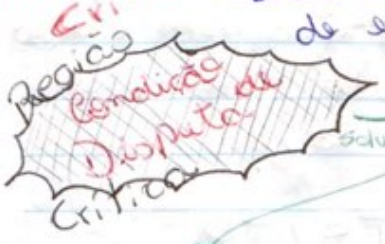
* Cuidado se for fazer thread com fork no meio do caminho

03/09/2019

Transparência: 05_Processos-Threads.pdf (de 24 até 37) FIM

lembrar das: Comunicação entre Processos
↳ comunicação é essencial

Regiões Críticas → acertar a sequência de eventos.



solução → Exclusão mútua.

Condições necessárias: 1 não pode estar junto
2 velocidade ✗
3 bloquear ✗
4 ~~espera eterna~~

Exclusão mútua ~~sem~~ espera ocupada (aluno chato) ▸ consome muito de recurso de troca de contexto maior
Exclusão mútua sem espera ocupada (aluno dominado)

Barreira → sincronização de blocos que não compartilham recursos
↳ MBI: essencial em memória distribuída (memória compartilhada: SEMÁFORO)

05/09/2019

Cache miss

Transparência: 06_Avaliacao-de-Programas-Paralelos.pdf

Escolher a ordem correta de operações resulta em um melhor aproveitamento dos recursos
ex: for(i→j→k) ou for(k→j→i)

Overhead de Paralelismo
↳ tempo para iniciar uma tarefa
↳ tempo para terminar uma tarefa
↳ sincronização (tempo do último) quando o último acaba
alguém que espera por alguém OCIOSO = overhead

*Leitura/Escrita no disco é IMPRECIZÁVEL (melhor deixar sequencial)

Tarefas pode ser imprevisíveis e isso torna difícil fazer o balanceamento } em paralelo pode receber uma tarefa mais fácil que ou, então ele acaba antes e fica OCIOSO

Eddington = Ociosidade
* falta de balanceamento de carga

Trivial: sem sincronização
↳ sem overhead

05/09/2019

Pegar um código sequência e otimizar ao máximo (TS)
Pegar um código paralelo (TP)

Pegar um programa feito em paralelo e executar em uma única thread isso não significa um código sequencial, ou seja, não pode ser usado para medir o TS. Deve ser feito um código do zero sequencial.

Aceleração (speedup): S(n) = TS/TP

sub-ótimo ↳ ↳ superlinear

Trivial = NÃO TEM OVERHEAD
Intensiva = COMUNICAÇÃO COM POUCOS (quanto mais, mais afastado do trivial).
Divide/Conquista = MAIS TEMPO PARA MANDAR DO QUE PROCESSAR
↳ muito tempo enviando do que executando

quão próximo de ser o speedup

número de processadores

Eficiência = (S(n)/n) * 100

Lei de Amdahl
Speedup = 1/(S + p/N)

Escalabilidade do algoritmo paralelo
↳ algoritmo pode suportar um aumento do tamanho do problema

Lei de Gustafson } Considera que o tempo de execução paralela é fixo
Speedup = S + Np

*escalabilidade mantendo

= Aceleração x Eficiência = programa ideal

10/09/2019

Transparência: 07_Modelo-Paralelo.pdf

DEBUG: serve para identificar erros

Profilers: serve para analisar o comportamento do código
↳ diminiz o algoritmo reduzindo o número de instruções
* ele não reduz, mas mostra os gargalos do programa
↳ gcc -Wall -pg prog.c prog

* redução de condicionais

*Pipeline (if): antes é feito o problema, depois é verificado se a condição foi atendida. Caso contrário, todo trabalho é perdido
↳ Exemplo clássico, slide 5

Tipos de paralelismo
↳ Paralelismo de dados
↳ Paralelismo funcional

Tipos de Otimização
GCC: O1
O2
O3
NÃO OTIMIZA ↳ O0

MPI: paralelização explícita (responsabilidade do programador)
API da linguagem: paralelização implícita (Open MP)
± Automatico: paralelização automática (loop padrão)

*Determinismo: programas não-determinísticos são mais complexos de depurar

OUTROS MODELOS

granularidade
flexibilidade
escalabilidade
modularidade
localidade

12/09/2019

Transparência: 08_PCAM.pdf slide 21

Particionamento
Comunicação
Aglomeração
Mapeamento

Decomposição do problema em subproblemas
DIVIDIR IGUALMENTE
Dados | Operações

Balanceamento de carga
Metis Moteur etc
comunicação custo > cargas iguais
melhor preencher todo espaço

Aglomerar o que se comunica: GRANULOSIDADE
Fina: muito pouco processado por comunicação
Grossa: muito computação por comunicação

Concorrência x Localidade (possibilidade de acesso ao CACHE)
MAPEAMENTO ESTÁTICO
MAPEAMENTO DINÂMICO

Quando é PRECISO
* Local x Global
* Estruturado x Estruturado
* Estática x Dinâmica
CONCORRENTES
um faz IO enquanto o outro faz CPU

Modelos de Aplicação

Workpool
Mestre Escravo
Dividir e Conquista
Pipeline
Fases Paralelas

17/09/2019

Transparência: 05_Maquinas-Paralelas.pdf (do 0 até 27) — 3 tipos
Cluster ; Grid ; Cloud Computing

Cray = máquinas vetoriais

Problema lei de Moore — Década 90

* MPP só é usado pelo Top500

Classifica os computadores mais poderosos
Top500 (atualmente o poder de processamento é NORMAL)

Cluster (CoPs, PoPs, COWs, CLUMPs)

Não é só para processamento paralelo, ela é paralela mas pode ser usada para resolver aplicações sequenciais.

Cluster Homogeneo → (mais simples de programar) → Cluster Heterogeneo

Grid
É um tipo de sistema paralelo e distribuído que permite o compartilhamento (Slide 16)

(Privada, Pública ou Híbrida)
Nuvem Computacional
Introduz uma nova forma de entrega de serviços de TI
Flexibilidade (quantidade necessária sempre disponível)
Elasticidade (Conteiner, ou seja... alterar o hardware rapidamente)
Economia
Virtualização
Pode ser acesso a máquina física

19/09/2019

Transparência: 05_Maquinas-Paralelas.pdf (de 27 até 35)

Coprocessadores de Aceleradores Gráficos ← um tipo de coprocessador

CPU = Específico (Por isso são tão bons)

CPU é genérico

Pipeline Gráfico
- Operações de vértices
- Primitivas de montagem
- Rasterização
- Operações de fragmentos
- Composição

time sharing (divisão de tempo)

maior quantidade de processos que eu pode processar em um ciclo

ALU = Unidade de processamento lógico (unidade lógica aritmética)

↳ Na GPU se você pensar apenas uma instrução sem mínimo X de threads são executadas; uma trabalho de X-1 dependendo energia.

Vetorizou o seu problema → USE → GPU

GPGPU (comércio)
OpenCL (código aberto)
CUDA (muito suporte "Nvidia")

19/09/2019

Transparência: 10_OpenMP.pdf

↳ API ñ é feito para memória distrib
ñ verifica dependências de deadlock

gcc main.c -fopenmp -o saida

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

#pragma exemplo slide 9

master faz uma coisa (trabalho diferente)
enquanto as threads fazem o mesmo trabalho

#pragma omp master exemplo slide 12

unicidade de execução

reduction(+:sum) = fazer uma zona crítica
(dentro depende da var) sub mut etc

scheduling exemplo slide 17

(o padrão é sempre esperar) nowait ñ precisa esperar pelas demais threads exemplo slide 18

sections barrier single critical ordered exemplo slide 20

OpenMP é apenas um facilitador na programação paralela. Em termos de visão do sistema operacional não existe diferença

include " " → procura na pasta do código
include < > → procura nas bibliotecas do SO

24/09/2019

Transparência: MPI_2018v02.pdf (Biblioteca)

~~PVM~~ extinto

Programação Paralela com Troca de Mensagens

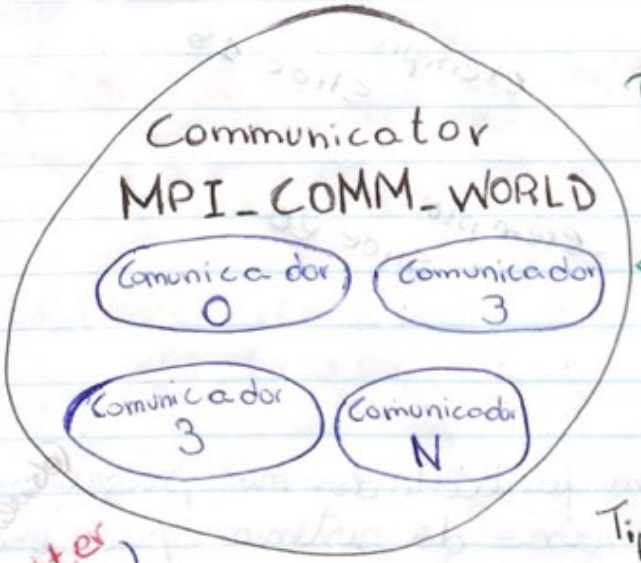
Descrição explícita
Troca de mensagens (send/recv) Quê? Quanto? Pra quem?
síncrono / assíncrono

SPMD
Single Program Multiple Data

MPMD
Multiple Program Multiple Data

MPI_INIT → geração dos processos
MPI_FINALIZE → se não usar o programa termina sem matar os processos
↳ vai sempre esperar o processo 0 e não o mestre.

Mestre
Escravo



Processos

Cada Comunicador possui seu próprio RANK

Scatter
Gather
↳ slide 45
↳ slide 42

Tipos básicos de C
=
Tipos de MPI

08/10/2019

Seminários Finais

10 min apresentação + 5 min arguição

BOINC ← TEMA
Computação voluntária

Set;@home

07/11/2019

Paralelismo na classificação de maçãs: Luciano

Sistemas de recomendação: Camila

Processamento de dados em tempo real para cidades inteligentes: Jean

Computação de Alto Desempenho: Ana

Paralelismo em fatoração de Matrizes para Sistemas de Recomendação: Franco

Aplicações de sistemas paralelos aos sistemas dinamicos: Délio

↳ Não apresentou, não fez o resumo e não fez o seminário. Todos os demais foram bem avaliados

Fim