PPT 1

- Computação Paralela:

-> Utiliza hardware para realizar computações mais rapidamente.

-> Foca-se na eficácia.

- Computação Concorrente:

-> Pode ou não fazer múltiplas execuções ao mesmo tempo.

PPT 2 e 3

- Programação Distribuída:

-> Clientes fazem pedidos aos servidores, para processar funções individuais.

-> Os servidores distribuem as solicitações por diversos processadores.

-> Os Sistemas têm SOs diferentes.

- Programação Paralela:

-> Vários PCs/Processadores cooperam para processar um problema dividido em subtarefas.

-> PCs usam SOs diferentes.

- Arquiteturas de Sistemas Paralelos:

1. Arquitetura - Processador e Tendências Lógicas

-> **Desenvolvimento Atual**: Processadores Multicore.

-> **Nº de Transístores**: Medida da complexidade e desempenho do processador.

-> **Lei de Moore**: O nº de transístores dobre a cada 18-24 meses.

-> **Razões para o desempenho não melhorar muito**: Taxa de relógio e melhorias na arquitetura.

2. Taxonomia de Flynn das Arquiteturas Paralelas

-> Caracterizar PCs paralelos pela organização do controlo global e os fluxos de dados e de controlo.

-> **Classificações de Flynn**:

---> SISD

---> MISD

---> SIMD (Passado e GPUs)

---> MIMD (Presente)

3. Organização da Memória nos PCs Paralelos

-> Os processadores ligam-se à memória global pela rede.

-> **Cache**: Memória Pequena, mas rápida, entre o proces. e a mem. central.

-> **Proces. Multicore**: Qlqr acesso de leitura retorna o valor escrito mais recente.

4. Paralelismo a Nível de Thread

-> **Hyperthreading**: É baseado numa duplicação da região do processador para o armaz. do estado do processador no chip dos processadores.

-> **Resultado**: Um proc. físico é visto como 2 lógicos (compartilhando recursos).

-> **Opções de desenho de projeto para Processadores Multicore**: Projetos hierárquicos, de Pipelines ou Baseados em Rede.

5. Redes de Interconexão: Redes de Comunicação de Computadores Paralelos

-> É usada para comunicação em sistemas multi-PCs.

-> **Topologia**:

---> **Redes Estáticas/Diretas**:

------> Os nós são interconectados ponto a ponto.

------> Diâmetro Pequeno: Pequenas distâncias de transmissão de mensagens.

------> Grau Pequeno de um Nó: Reduz requisitos de hardware.

------> Alta Bisseção de Banda Larga: Ajuda a alcançar alto rendimento de dados.

------> Alta Conectividade: Aumenta a fiabilidade.

--> **Redes Dinâmicas/Indiretas**:

------> Os nós são conectados indiretamente, por vários switches intermediários.

------> Os switches são configurados dinamicamente, de acordo com os requisitos da transmissão da mensagem.

-> **Técnicas Encaminhamento**:

---> Algor. de Encaminhamento: Seleção do caminho.

---> Estratégia de Comutação: Modo de Transmissão.

6. Encaminhamento e Switching (Chaveamento)

-> **Algoritmo de Roteamento**: Encontra um caminho na rede pelo qual uma mensagem deve ser enviada, segundo certas condições.

-> Estratégia de **Switching**: Define como é que uma mensagem é enviada sobre um caminho escolhido pelo algor.

7. Caches e Hierarquia de Memória

-> Localidade de Acesso à Memória:

---> **Espacial**: Acesso vizinho aos locais de memória principal em pontos consecutivos, durante a execução do programa.

---> **Temporal**: O mesmo local de memória é acedido várias vezes em pontos consecutivos, durante a execução do programa.

PPT 4 e 5

- Modelos para Sist. Paralelos:

-> **Mod. Máquinas Paralelas**: O nível mais baixo da Abstração, descrição do hardware do sistema.

-> **Mod. Arquiteturas Paralelas**: Abstração dos modelos de máquinas, SIMD ou MIMD, organização de memória.

-> **Mod. Computacionais Paralelos**: Extensão dos modelos arquiteturais que permitem construir algoritmos para que os custos do modelo possam vir a ser corretamente considerados.

-> **Mod. Programação Paralela**: Descrição de um sistema pela descrição da linguagem de programação e pelo seu ambiente.

- Paralelização de Programas:

1. Decomposição d'Computações;
2. Atribuição das Tarefas a Processos (Escalonamento);
3. Orquestração e Mapeamento dos Processos aos Cores e/ou Processadores Físicos.

- Níveis de Paralelismo:

1. Paralelismo de Instruções;
2. Paralelismo de Dados;
3. Ciclos (Loops) Paralelos:

3.1. Loop Distribuído: As instruções sem interdependência são separadas e podem ser paralelizadas.

3.2. Paralelismo DOALL: Não há dependências e as instruções de um loop podem ser executadas paralelamente.

4. Paralelismo de Funções;
5. **Padrões** de Programação / Desenho Paralelo: Estruturas de Coordenação das Threads:

5.1. **Criação de Threads**:

5.1.1. Estática: Um nº fixo de Threads é criado no início da execução, e são destruídos no fim da execução.

5.1.2. Dinâmica: A criação de Threads é feita sempre que necessário.

5.2. **Fork-Join**:

5.2.1. Uma Thread existente cria uma 2ª Thread; o encadeamento é arbitrário e diferentes linguagens e ambientes de programação exibem diversas características.

5.3. **Parbegin-Parend**:

5.3.1. Criação e destruição simultânea de várias Threads.

5.3.2. As instruções no bloco são mapeadas para Threads diferentes.

5.4. **SPMD e SIMD**:

5.4.1. Todas as Threads executam o mesmo programa com dados diferentes.

5.4.2. SPMD: Num certo momento, Threads diferentes executam instruções diferentes (assíncrono).

5.4.3. SIMD: A mesma instrução é executada simultaneamente por todas as Threads (síncrono).

5.5. **Master-Slave**:

5.5.1. Uma única Thread controla todas as computações de um programa.

5.5.2. Cria Threads slaves e atribui-lhes computações.

5.6. **Modelo Cliente-Servidor**:

5.6.1. Várias Threads clientes fazem solicitações à Thread servidor, a qual atende os clientes de forma paralela.

5.6.2. Extensões: Threads que são clientes e servidores, same time.

5.7. **Pipelining**:

5.7.1. Permite paralelismo, apesar das dependências dos dados.

5.8. **Pool de Tasks**:

5.8.1. Estrutura de dados que gere partes do programa, como tasks.

5.9. **Thr. Produtor-Consumidor**:

5.9.1. As Threads produtoras criam os dados, e as Threads de consumo usam esses dados.

- Distribuição de Dados para Vetores e Matrizes:

-> Os dados são particionados em conjuntos menores, distribuídos pelas cores ou processadores para usar na execução.

-> **Memória Distribuída**: Os dados atribuídos a 1 proces. ficam guardados na memória local e só podem ser acedidos por esse processador.

-> **Memória Partilhada**: Os dados são guardados na mesma memória partilhada e cada core acede a dados diferentes de acordo com o padrão de distribuição dos dados, o qual tem d'evitar conflitos de acesso.

- Troca de Informações:

-> Memória Distribuída:

---> Operações de comunicação que permitem a troca de informação pelo envia de mensagens;

---> Distinção de comunicação ponto-a-ponto e global.

-> Memória Partilhada:

---> Uso de variáveis partilhadas, com acesso protegido por sincronização.

- Processos e Threads:

-> **Processos**:

---> Programa em execução, com o código e toda a info.;

---> Tem um espaço de endereçamento próprio.

-> **Threads**:

---> É uma extensão do modelo de processo;

---> Partilham o espaço de endereçamento do seu processo;

---> A sua criação é rápida.

-> **Problemas e Soluções**:

---> **Correção em Computação Paralela**:

------> Critério 1: O output deve ser sempre o mesmo

------> Critério 2: O output deve ser o mesmo se o código não for executado em paralelo.

---> **Condições de Corrida**;

---> **Estado Partilhado**:

------> Problema do Banco: O estado partilhado entre múltiplas Threads cria problemas que um ambiente single-Thread não tem.

---> **Locks e Semáforos**:

------> Var. Partilhadas podem ser usadas como sinais de que todas as execuções entendem e respeitam, coordenando-se por passagem de mensagens.

------> Locks/Mutexes: Objetos partilhados usados para sinalizar que um estado partilhado está a ser lido ou modificado.

------> Semáforos: São sinais usados para proteger o acesso a recursos limitados.

---> **Sync. por Barreiras**;

---> **Variáveis de Condição**:

------> 1 Processo pode usar esta var. para sinalizar que acabou a sua vez, para outros avançarem.

---> **Deadlocks**:

------> Situação em que 2 ou mais processos ficam bloqueados, cada um à espera que os outros terminem.

PPT 4 e 5 - Complementar

- Modelos de Programação Paralela:

-> **Memória Distribuída**: A aplicação consiste numa coleção de processos diferentes.

-> **Memória Partilhada**: A aplicação é uma coleção de Threads, que podem ser criadas dinamicamente, ou manualmente.

-> **Dados Paralelos**: Um único Thread de operações paralelas, que não se adapta a todos os problemas.

-> **Híbridos**: MPI no topo da hierarquia, com memória partilhada pelos Threads em cada máquina.

- Programação em Memória Distribuída:

-> Prog. com OpenMPI:

---> Não há var. partilhadas;

---> Os programas executam semelhantemente a programas sequenciais uniprocesso;

---> **Conceitos MPI**:

------> **Grupo**: Ajuntamento de processos.

------> **Contexto**: Onde é enviada uma mensagem.

------> **Comunicador**: Um grupo e um contexto.

------> **Rank**: Identificador de um processo.

------> **Tag**: Identificador de uma mensagem.

---> **Coordenação com o Ambiente**:

MPI_Comm_size: Nº de processos a correr;

MPI_Comm_rank: Informa o rank de um processo;

MPI_Comm_WORLD: Designa todos os processos em execução.

- Programação em Memória Partilhada:

-> Prog. com Threads:

---> Padrão POSIX, portável, mas pesado e lento.

---> Var. Partilhadas: Globais.

---> Var. Privadas: Locais.

-> Prog. com OpenMP:

---> Suporta a programação científica.

---> Alternativa às Threads.

---> Especificação aberta para processamento paralelo.

---> Var. Partilhadas: Shared.

---> Var. Privadas: Private.

---> **Permite**:

------> Separar o programa em regiões sequenciais e paral.

------> Confiar na gestão automática das necessidades de memória.

------> Usar os mecanismos de sincronização já se encontram disponíveis.

---> **Não Permite**:
------> Paralelismo Auto.;
------> Garantias de melhor Desempenho;
------> Evitar por completo inconsistências no acesso partilhado aos dados.
---> **Sincronização**:
------> **Secções Críticas**:
#pragma omp critical
------> **Diretivas Barrier**:
#pragma omp barrier
------> **Func. d'Lock Explícitas**:
omp_set_lock(l1);
omp_unset_lock(l1);
------> **Regiões de Thread Único dentro de Regiões Paralelas**:
#pragma omp single

PPT 6

- Estilos de Arquitetura:
-> **Conector**: Mecanismo que media a comunicação, coordenação ou cooperação entre componentes.

-> **Estilos**:
---> **Arquitetura em Camadas**:
------> Camada de Interface de App: Contém unidades para interagir com os utilizadores / apps externas.
------> Camada de Processamento: Contém as funções de uma aplicação.
------> Camada de Dados: Contém os dados que um cliente quer manipular pelos componentes da app.
---> **Arquitetura Baseada em Objetos e Serviços**:
------> Essência: Os componentes são objetos, ligados entre si através de chamadas de procedimentos.
------> Encapsulação: Diz-se que os objetos encapsulam dados e oferecem métodos sobre eles.
---> **Arquitetura Baseada em Recursos**:
------> **Arquitetura RESTful**:
1. Essência: O sistema distribuído é uma coleção de recursos, e cada recurso é gerido por componentes.
2. Operações Básicas: PUT, GET, DELETE e POST.
------> **SOAP vs. RESTful**:
1. Muitos preferem o RESTful por a interface de um serviço ser mais simples.
2. O lado menos positivo é que muito precisa de ser feito no espaço dos parâmetros.
---> **Arquitetura Editor-Assinante**:
------> **Acoplamento Temporal e Referencial**:

	T.A.	T.D.
R.A.	Ligação Direta	Caixa de Correio
R.D.	Baseada Em Eventos	Espaço de Dados Partilhado

T.A. – Temporalmente Acoplado
R.A. – Ref. Acoplado
T.D. – Temp. Desacoplado
R.D. – Ref. Desacoplado

- Organização do Middleware:
-> Utilização de Entidades Legadas para Construir Middleware:
---> Problema: As interfaces oferecidas por um componente legado provavelmente não são adequadas para todas as apps.
---> Solução: Um Wrapper / Adaptador oferece uma interface aceitável para uma app do cliente.
-> Organização dos Wrappers:
---> Duas soluções: 1-para-1 ou através de um broker.
-> Desenvolvimento de Middleware Adaptável:
---> Problema: O conceito de Middleware implica em soluções que são boas para a maior parte de apps; adapta-se.

- Organizações Centralizadas:
-> Modelo Básico Client-Server:
---> **Servidor**: Processo que oferece serviços.

---> **Cliente**: Processo que usa serviços.
-> Arquiteturas de Sistemas Multiníveis:
---> **Nível Único**: Configuração de terminal/PC centra.
---> **2 Níveis**: Configuração cliente/servidor único.
---> **3 Níveis**: Cada camada numa máquina separada.

- Arquitetura Descentralizadas:
-> Organizações Alternativas:
---> **Distribuição Vertical**: Vem da divisão de apps em 3 lvls lógicos, e da execução dos componentes de cada lvl num server diferente.
---> **Distribuição Horizontal**: Um cliente ou server pode estar fisicamente dividido em partes logicamente equivalentes.
---> **Arquiteturas P2P**: Os processos são todos iguais: as funções que precisam de ser executadas são representadas por todos os processos.
-> P2P Estruturado: A estrutura lógica é organizada numa topologia específica, e o protocolo garante que qlqr nó pode pesquisar pela rede.
-> P2P Não Estruturado:
---> Essência: Cada nó tem uma lista ad hoc de vizinhos. A estrutura lógica resultante é parecida a um grafo aleatório.
---> Pesquisa na Rede:
------> **Inundação (Flooding):** O nó emissor passa o pedido pelo recurso a todos os vizinhos.
------> **Random Walk:** O nó emissor passa o pedido pelo recurso a 1 vizinho aleatório.
------> A R.W. é mais eficiente, mas também pode ser + lenta.

- Modelos Híbridos:
-> **Para Redes P2P**:
---> Essência: São uma combinação de modelos P2P e client-server, com um server central para ajudar os pares a encontrarem-se.
-> **Arquitetura de Edge Computing**:
---> Essência: É aquela na qual o processamento acontece no local físico do cliente / fonte de dados.

PPT 7

- Socket:
-> End Point da comunicação.
-> Envio de mensagens entre processos via rede.
-> É uma API que dá suporte à criação de apps de rede.
-> UDP:
---> Coleção de Mensagens;
---> Sem garantias, best esforço;
---> Sem estabelecer uma sessão.
-> TCP:
---> Fluxo de bytes;
---> Garantias de entrega e integridade;
---> Estabelece uma sessão subjacente ao tráfego de dados.
-> Caracterização de um Socket:
---> Protocolo de Comunicação (TCP ou UDP);
---> Endereço IP (hospedeiro);
---> Nº da Porta (processo).

- Clientes e Servidores:
-> Programa Cliente:
---> Inicia a comunicação;
---> Tem de saber o IP e porta do servidor;
---> Solicita um serviço.
-> Programa Servidor:
---> Aguarda ligações;
---> Obtém o IP e porta do cliente na ligação;
---> Fornece o serviço.

- Ambiente Tradicional de RPC vs. Java RMI:
-> **RPC**: Remote Procedure Call:
---> Chamada provedoral de um processo de uma máquina servidora, a partir de um processo numa máquina cliente.
---> **RMI**: Remote Method Invocation:
------> RPC para o ambiente orientado a objetos.
------> **RMI – Stub** (Cliente): Transforma os parâmetros em formato independente de máquina (mashalling).
------> **RMI – Esqueleto** (Servidor): Recebe a requisição com o nome do método, decodifica (unmarshals) os parâmetros e utiliza-os para chamar o método no objeto remoto.

PPT 8

- Protocolo HTTP:
-> Funciona como um protocolo de pedido-resposta, e é um protocolo muito simples para trocar mensagens de texto entre 2 máquinas.

- HTTP 0.9:
-> **Objetivos**:
---> Permitir transferência de TXT;
---> Permitir pesquisas em ficheiros de hipertexto;
---> Permitir negociação de formatos de ficheiros;
---> Permitir referenciar outros servers a clientes.
-> **Funcionalidades**:
---> Resposta em HTML;
---> A ligação termina após a resp.

- HTTP 1.0:
-> Nunca chegou a ser um standard;
-> Métodos: GET, HEAD e POST.

- HTTP 1.1:
-> **Novos Métodos**: PUT (novo conteúdo, ou modificação do existente), DELETE, TRACE, OPTIONS, PATCH e CONNECT.
-> **Limitações**:
---> Os pedidos e respostas são sequenciais e para haver paralelismo tem de haver múltiplas ligações ao mesmo servidor.

- HTTP 2.0:
-> Surgiu a partir do protocolo experimental SPDY da Google, pretendendo melhorar a latência do anterior, não requerer múltiplas ligações paralelas, entre outros.
-> **Ligações**: Todas as comunicações são feitas por TCP; uma stream é um canal virtual com um identificador…
-> **Multiplexagem**: Permite enviar mensagens em várias streams, paralelamente.
-> Ainda está a ser adotado mundialmente.

- HTTP/3:
-> Será baseado no protocolo experimental da Google (QUIC);
-> Utilizará o protocolo UDP;
-> Já está em criação, apesar do HTTP/2 ser mundialmente usado por 34% das pessoas.

PPT 9

- Nomes:
-> São usados para identificar entidades, referir localizações e partilhar recursos.
-> Nomes N – 1 Recursos.

- Endereços:
-> Um recurso pode ter um ou mais APs/Endereços, que podem ser IPs, MACs ou de Memória.

- Resolução de Nomes em Sistemas:
-> **Nomeação Plana**: Sistemas onde identificadores são nomes não-estruturados.
---> **Soluções Simples**:
------> **Broadcast**: Pedidos de resolução são enviados para todos os nós da rede.
------> **Multicast**: Os pedidos são enviados apenas para nós pertencentes ao grupo.
---> **Tabelas de Dispersão Distribuídas**:
------> Estrutura que associa chaves de pesquisa a valores.
------> Algoritmo Baseado nas Tabelas: Cada nó tem um nó sucessor e um predecessor.
------> Para resolver um nome: O nó inicial verifica se sabe o endereço do destinatário, e senão, passa aos sucessores.
-> Nomeação Estruturada:
---> Nomes não estruturados tendem a ser inconvenientes para redes complexas.
---> Nomes estruturados são organizados num namespace.
---> **Namespace**: Grupo de componentes do nome num formato tipo árvore:
------> **Raiz**: Nome + Alto Nível;
------> **Diretórios**: Subdivisão;
------> **Folhas**: Recursos.
---> **Distribuição de Nomes**:
------> Dividido em 3 camadas **lógicas**: **Global** (alto nível), **Administrativa** (nós intermédios), **Final** (recursos).
---> **Resolução de Nomes Estruturados**: **Iterativamente** (o server resolve o que der e devolve ao cliente, e repete) e **recursivamente** (o server resolve e passa ao próximo server, até terminar).
-> **Nomeação Baseada em Atributos**:
---> Em certos casos, pode-se querer indexar os nomes dos recursos por atributos.

PPT 10

- **Webservices**: Serviços cridos de modo a suportar a interação entre máq. numa rede.
- **Funcionamento**: A app solicita uma das operações disponíveis no webservice, e este efetua o processamento dos dados e envia-os.
- Tecnologias Mais Usadas:
-> **AJAX**: Permite aos browsers obter info de um server de forma assíncrona, sem reler a página toda.
-> **SOAP**: Protocolo de mensagens que usa XML para trocar info com um server.
-> **REST**: Arquitetura para interoperabilidade entre PCs na internet; os servers expõem serviços através de recursos na web; é mais leve.

Perguntas

- **Arq. P2P**: Processos iguais, funcionando como cliente e servidor ao mesmo tempo.
- **RMI**: Estilo baseado em objs.
- **TCP/IP**: Arq. Em camadas.
- **Navegador e Servidor WEB**: Arq. Client-Server.
- Utilização de verbos **HTTP** para operações básicas: **REST**.
- **Modelo de referência OSI**: Camadas.
- **Flooding**: Faster, - eficiente.
- **RW**: + lento + eficiente.
- Um **conector** provê mecanismos de comunicação, coordenação ou cooperação para: Chamada de procedimentos remotos; Streaming de dados entre processos; Passagem de mensagens entre processos.
- **Multiprocessing**: Process. Coarse-Grained | CPU-Bound.
- **Threading**: Processamento Fine-Grained | IO-Bound.
- pthread_create retorna 0 no sucesso na criação da Thread.
- **Granularidade**: Paralelismo **Instruction-Lvl** (small blocks); **Task-Lvl** (multi-threading); **Bit-Lvl** (menos instruções).

Multithreading: **Coarse-Grained** -> Alterna nas maiores interrupções; **Fine-Grained** -> Alterna após cada instrução e oculta a demora no acesso à memória.