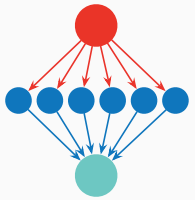


Por que programação paralela?

Processamento Paralelo

Prof. Oberlan Romão
Departamento de Computação e Eletrônica – DCEL
Centro Universitário Norte do Espírito Santo – CEUNES
Universidade Federal do Espírito Santo

Perspectiva do programador sobre desempenho

Como você faz seu programa rodar mais rápido?

Perspectiva do programador sobre desempenho

Como você faz seu programa rodar mais rápido?

- Resposta antes de 2004:
  - Apenas espere 6 meses e compre um computador novo!
  - Ou se você está realmente obcecado, você pode aprender sobre paralelismo.

Perspectiva do programador sobre desempenho

Como você faz seu programa rodar mais rápido?

- Resposta antes de 2004:
  - Apenas espere 6 meses e compre um computador novo!
  - Ou se você está realmente obcecado, você pode aprender sobre paralelismo.
- Resposta após 2004:
  - Você precisa escrever software paralelo!

Por que programação paralela?

Por que programação paralela?

Tempos de mudança

- De 1986 a 2004, os microprocessadores foram acelerando como um "foguete", aumentando desempenho em uma média de 50% ao ano
- Desde então, caiu para cerca de 20% o aumento por ano

Por que programação paralela?

Uma solução inteligente

- Em vez de projetar e construir microprocessadores mais rápidos, colocar vários processadores em um único circuito integrado

Por que programação paralela?

Agora cabe aos programadores

- Adicionar mais processadores não ajuda muito se os programadores não estiverem cientes deles...
- ... ou se não sabem como usá-los de forma eficiente
- Os programas sequenciais não se beneficiam dessa abordagem (na maioria dos casos)

Por que programação paralela?

Por que precisamos de um aumento crescente desempenho?

- O poder computacional está aumentando, assim como nossos problemas e necessidades de computação
- Problemas com os quais nunca sonhamos foram resolvidos devido o aumento do poder computacional
  - Por exemplo, a decodificação do genoma humano
- Problemas mais complexos ainda estão esperando para serem resolvidos

Por que programação paralela?

- Dois dos principais motivos para utilizar programação paralela são:
  - Reduzir o tempo necessário para solucionar um problema
  - Resolver problemas mais complexos e de maior dimensão
- Outros motivos são:
  - Tirar proveito de recursos computacionais não disponíveis localmente ou subaproveitados
  - Ultrapassar limitações de memória quando a memória disponível em um único computador é insuficiente para a resolução do problema
  - Ultrapassar os limites físicos de velocidade que atualmente começam a restringir a possibilidade de construção de computadores sequenciais cada vez mais rápidos
  - Fornecer concorrência (fazer múltiplas coisas ao mesmo tempo)
  - Economia de tempo e custo

Por que programação paralela?

Economia de tempo e custo

- Em teoria, lançar mais recursos em uma tarefa reduzirá seu tempo até a conclusão, com potencial economia de custos
- Computadores paralelos podem ser construídos a partir de componentes baratos e básicos


Por que programação paralela?

Resolver problemas maiores / mais complexos

- Muitos problemas são tão grandes e/ou complexos que é impraticável ou impossível resolvê-los em um único computador, especialmente com a memória limitada do computador
  - "Grand Challenge Problems" requerem PetaFLOPS e PetaBytes de recursos computacionais.
  - Buscadores online
  - Controle de transações bancarias (milhões a cada segundo)

Por que programação paralela?

Fornecer concorrência

- Um computador, com um único processador, só pode fazer uma coisa de cada vez. Múltiplos computadores podem fazer muitas coisas simultaneamente.

Por que programação paralela?

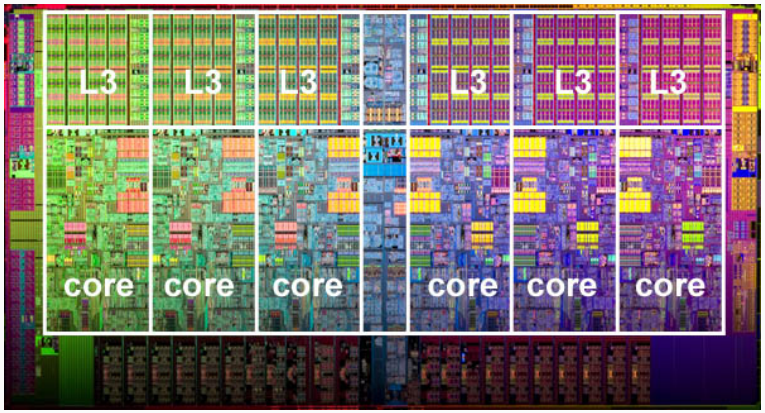
Tirar proveito dos recursos não-locais

- Usando recursos de computação em uma rede, ou mesmo na Internet, quando os recursos locais do computador são escassos ou insuficientes

Por que programação paralela?

Melhor utilização do hardware paralelo

- Computadores modernos, mesmo smartphones, possuem arquiteturas paralelas com múltiplos processadores/núcleos
- Aplicações paralelas são especificamente destinadas a hardware paralelo com múltiplos núcleos, threads, etc.
- Na maioria dos casos, programas sequenciais rodando em computadores modernos "desperdiçam" o poder potencial de computação



Processador Intel Xeon com 6 cores e 6 unidades de cache L3

Por que programação paralela?

- Tradicionalmente, a programação paralela foi motivada pela resolução ou simulação de problemas fundamentais da ciência/engenharia de grande relevância científica e econômica, denominados como Grand Challenge Problems (GCPs)
- Tipicamente, os GCPs simulam fenômenos que não podem ser medidos por experimentação:
  - Fenômenos climáticos: movimento das placas tectônicas
  - Fenômenos físicos: órbita dos planetas
  - Fenômenos químicos: reações nucleares
  - Fenômenos biológicos: genoma humano
  - Fenômenos geológicos: atividade sísmica
  - Componentes mecânicos: aerodinâmica/resistência de materiais em naves espaciais
  - Circuitos eletrônicos: verificação de placas de computador
  - ...

Por que programação paralela?

- Atualmente, 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*)
  - Exploração de petróleo
  - Serviços de busca online
  - Serviços associados a tecnologias multimídia e telecomunicações
  - Computação gráfica e realidade virtual
  - Diagnóstico médico assistido por computador
  - ...

Em 1998/1999...

Intel Pentium III

- Número de núcleos: 1
- Litografia: 250nm
- Cache: 512Kb
- Frequência máxima: 450MHz

Em 2008...

Intel Core i7-940

- Número de núcleos: 4
- Litografia: 45nm
- Cache: 4Mb
- Frequência máxima: 3,20GHz

Em 2018...

Intel Core i9-7980XE

- Número de núcleos: 18
- Litografia: 14nm
- Cache: 24Mb
- Frequência máxima: 4,20GHz

Em 2018...

Intel Xeon Phi Processor 7295

- Número de núcleos: 72
- Litografia: 14nm
- Cache: 36Mb
- Frequência máxima: 1,60GHz

Top5 - Novembro de 2018

Top5 - Novembro de 2018

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/SC/Oak Ridge National Laboratory United States	2,397,824	143,500.0	200,794.9	9,783
2	**Sierra** - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
3	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
4	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 , NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482
5	**Piz Daint** - Cray XC50, Xeon E5-2690v3 12C 2.6GHz, Aries interconnect , NVIDIA Tesla P100 , Cray Inc. Swiss National Supercomputing Centre (CSCS) Switzerland	387,872	21,230.0	27,154.3	2,384

www.top500.org

Summit - IBM Power System

Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband

Site:	DOE/SC/Oak Ridge National Laboratory
System URL:	http://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/
Manufacturer:	IBM
Cores:	2,397,824
Memory:	2,801,664 GB
Processor:	IBM POWER9 22C 3.07GHz
Interconnect:	Dual-rail Mellanox EDR Infiniband
Performance	
Linpack Performance (Rmax)	143,500 TFlop/s
Theoretical Peak (Rpeak)	200,795 TFlop/s
Nmax	16,693,248
HPCG [TFlop/s]	2,925.75
Power Consumption	
Power:	9,783.00 kW (Submitted)
Power Measurement Level:	3
Measured Cores:	2,397,824
Software	
Operating System:	RHEL 7.4

Sunway TaihuLight

Sunway TaihuLight - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway

Site:	National Supercomputing Center in Wuxi
Manufacturer:	NRCPC
Cores:	10,649,600
Memory:	1,310,720 GB
Processor:	Sunway SW26010 260C 1.45GHz
Interconnect:	Sunway
Performance	
Linpack Performance (Rmax)	93,014.6 TFlop/s
Theoretical Peak (Rpeak)	125,436 TFlop/s
Nmax	12,288,000
HPCG [TFlop/s]	480.8
Power Consumption	
Power:	15,371.00 kW (Submitted)
Power Measurement Level:	2
Software	
Operating System:	Sunway RaiseOS 2.0.5

Programa sequencial vs. paralelo

Programa sequencial

- Uma série de ações que produz um resultado
- Contém um único thread de controle
- É executado por um único processo

Programa sequencial vs. paralelo

Programa sequencial

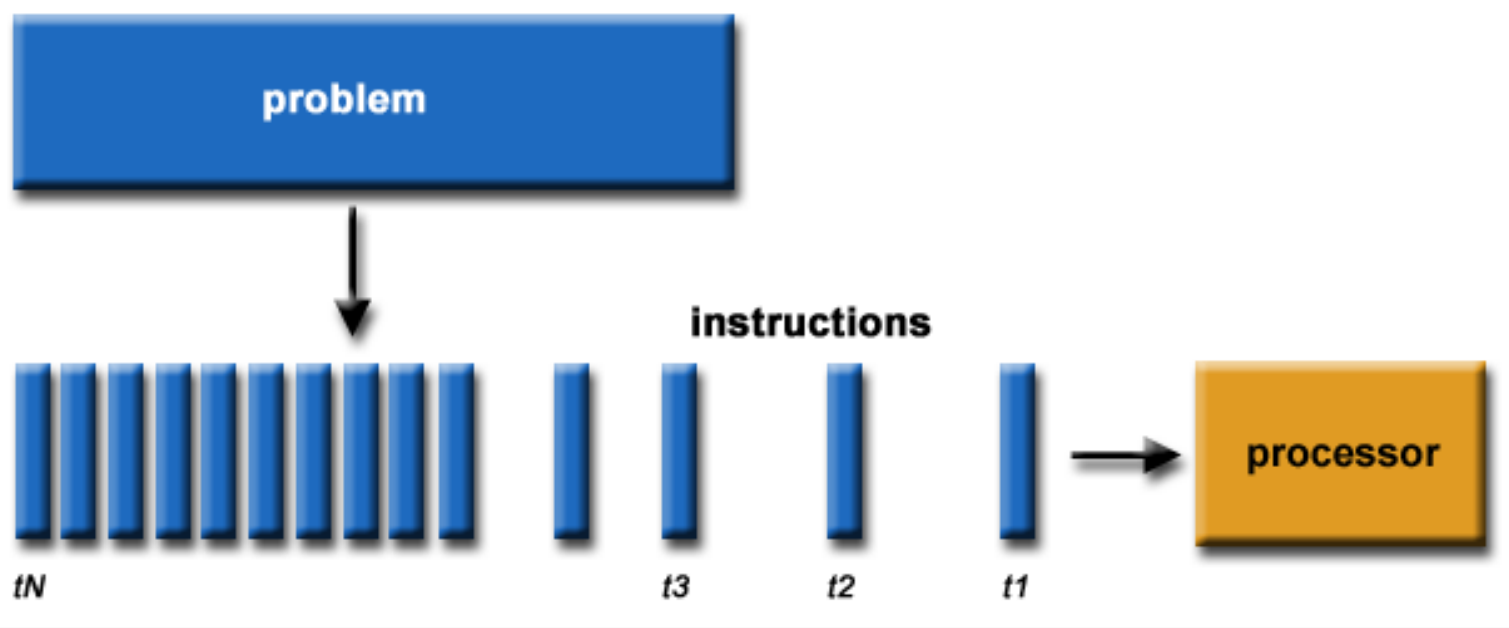
- Uma série de ações que produz um resultado
- Contém um único thread de controle
- É executado por um único processo

Programa paralelo

- Contém dois ou mais *threads* ou processos que cooperam entre si para resolver um problema
- Cada um executa um programa sequencial (tem seu próprio estado)
- Se comunicam via memória compartilhada ou troca de mensagens
- Podem ser executados concorrentemente em um único processador ou em paralelo em um chip com vários núcleos ou em vários computadores.

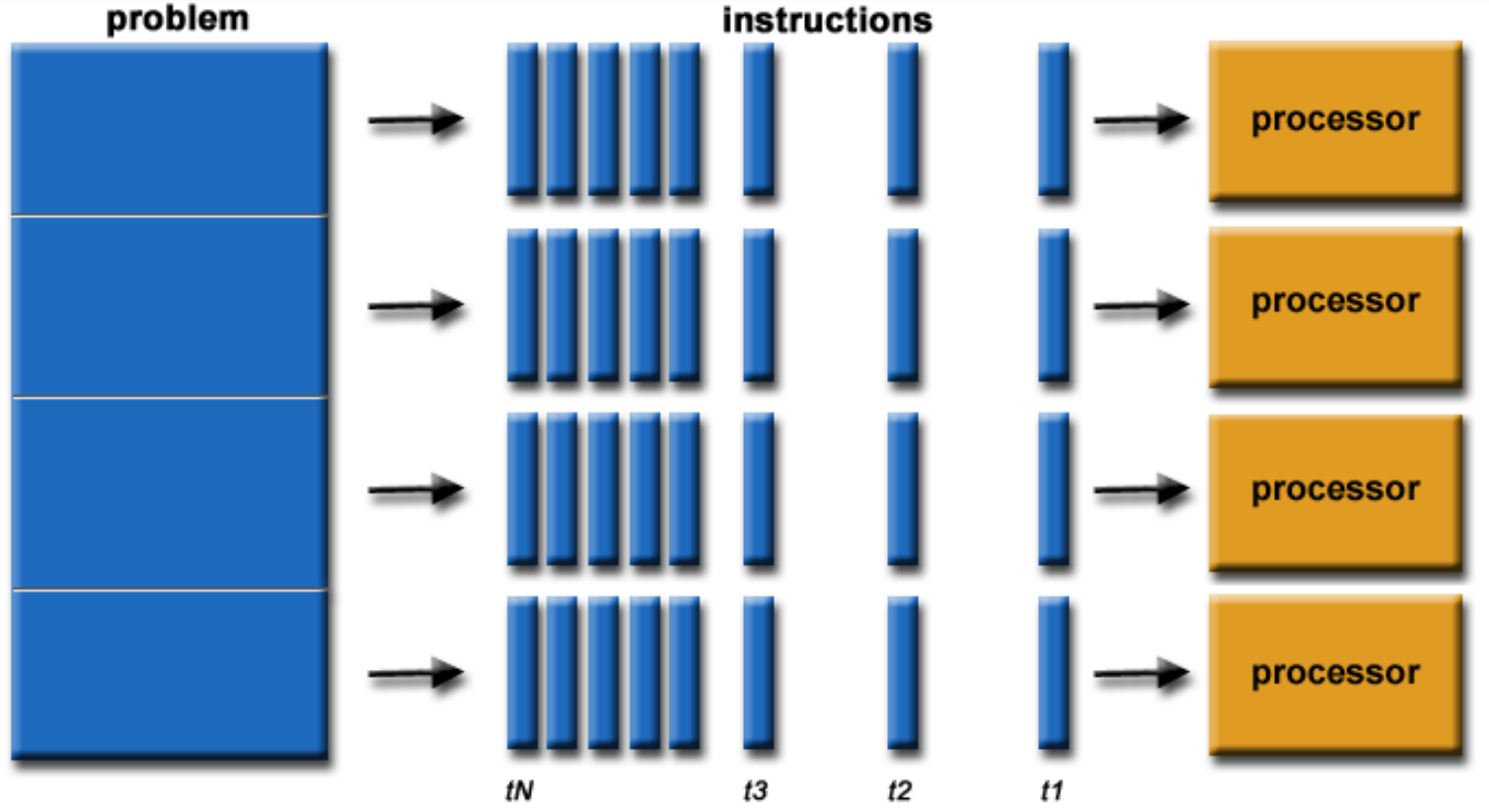
Programa sequencial

- Um programa é considerado sequencial quando este é visto como uma série de instruções sequenciais que devem ser executadas em um único processador



Programa paralelo

- Um programa é considerado paralelo 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



Terminologia

Computação concorrente

- Vários processos (*threads*) podem estar em progresso
- Processos/*threads* compartilham a CPU
- Modelo de memória compartilhada

Terminologia

Computação concorrente

- Vários processos (*threads*) podem estar em progresso
- Processos/*threads* compartilham a CPU
- Modelo de memória compartilhada

Computação paralela

- Mais de uma tarefa pode ser executada simultaneamente
- Processos concorrentes executam em processadores/núcleos distintos
- Modelo de memória compartilhada ou troca de mensagens

Terminologia

Computação concorrente

- Vários processos (*threads*) podem estar em progresso
- Processos/*threads* compartilham a CPU
- Modelo de memória compartilhada

Computação paralela

- Mais de uma tarefa pode ser executada simultaneamente
- Processos concorrentes executam em processadores/núcleos distintos
- Modelo de memória compartilhada ou troca de mensagens

Computação distribuída

- Processos distribuídos entre máquinas que se comunicam através de uma rede
- Modelo de troca de mensagens

Paralelização automática(?)

Não podemos simplesmente deixar tudo por conta do compilador/hardware?

Paralelização automática(?)

Não podemos simplesmente deixar tudo por conta do compilador/hardware?

- Os compiladores não podem inventar um algoritmo diferente
- Acredita-se que o paralelismo de hardware não vá muito mais longe

Paralelização via Compiladores

Depois de 30 anos de pesquisas intensivas

- Apenas sucesso limitado na detecção de paralelismo e transformações de programas
  - Paralelismo em *nível de instrução pode ser detectado*
  - Paralelismo em *loops aninhados com expressões de índice simples podem ser analisados*
  - Técnicas de análise, tais como análise de dependência de dados, análise de ponteiros, análise de fluxo, etc, quando aplicadas à programas, muitas vezes, demoram muito tempo e tendem a ser frágeis, ou seja, podem falhar após pequenas modificações no programa

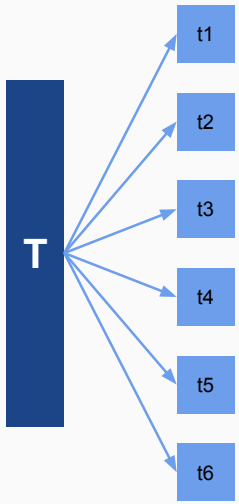
Em vez de "treinar" os compiladores a reconhecer partes paralelizáveis, os programadores têm sido treinados para escreverem seus códigos paralelos

Como paralelizar?

- Três passos:

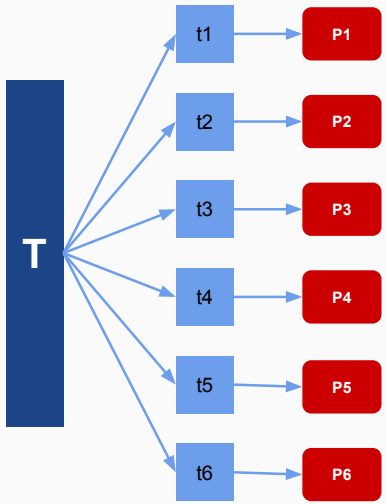
Como paralelizar?

- Três passos:
  1. Quebrar a tarefa em tarefas menores



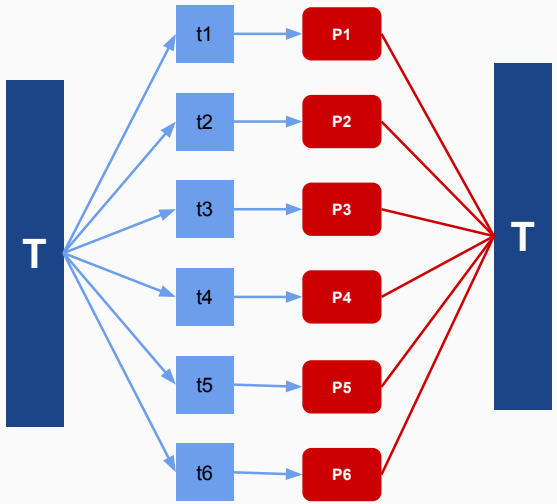
Como paralelizar?

- Três passos:
  1. Quebrar a tarefa em tarefas menores
  2. Atribuir as tarefas menores a vários processadores/núcleos para serem processados simultaneamente



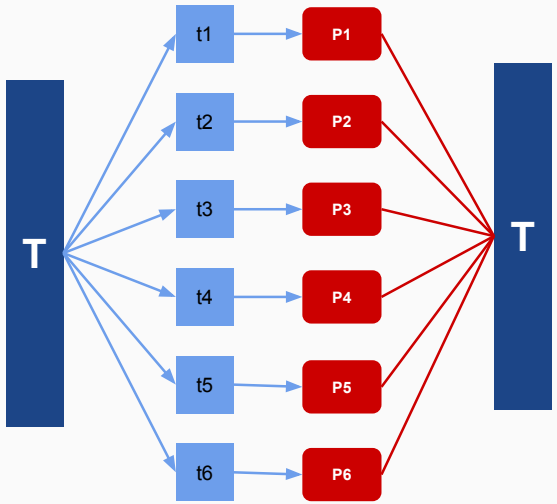
Como paralelizar?

- Três passos:
  1. Quebrar a tarefa em tarefas menores
  2. Atribuir as tarefas menores a vários processadores/núcleos para serem processados simultaneamente
  3. Coordenar a comunicação/sincronização entre os processadores/núcleos



Como paralelizar?

- Três passos:
  1. Quebrar a tarefa em tarefas menores
  2. Atribuir as tarefas menores a vários processadores/núcleos para serem processados simultaneamente
  3. Coordenar a comunicação/sincronização entre os processadores/núcleos
- Parece simples, não?

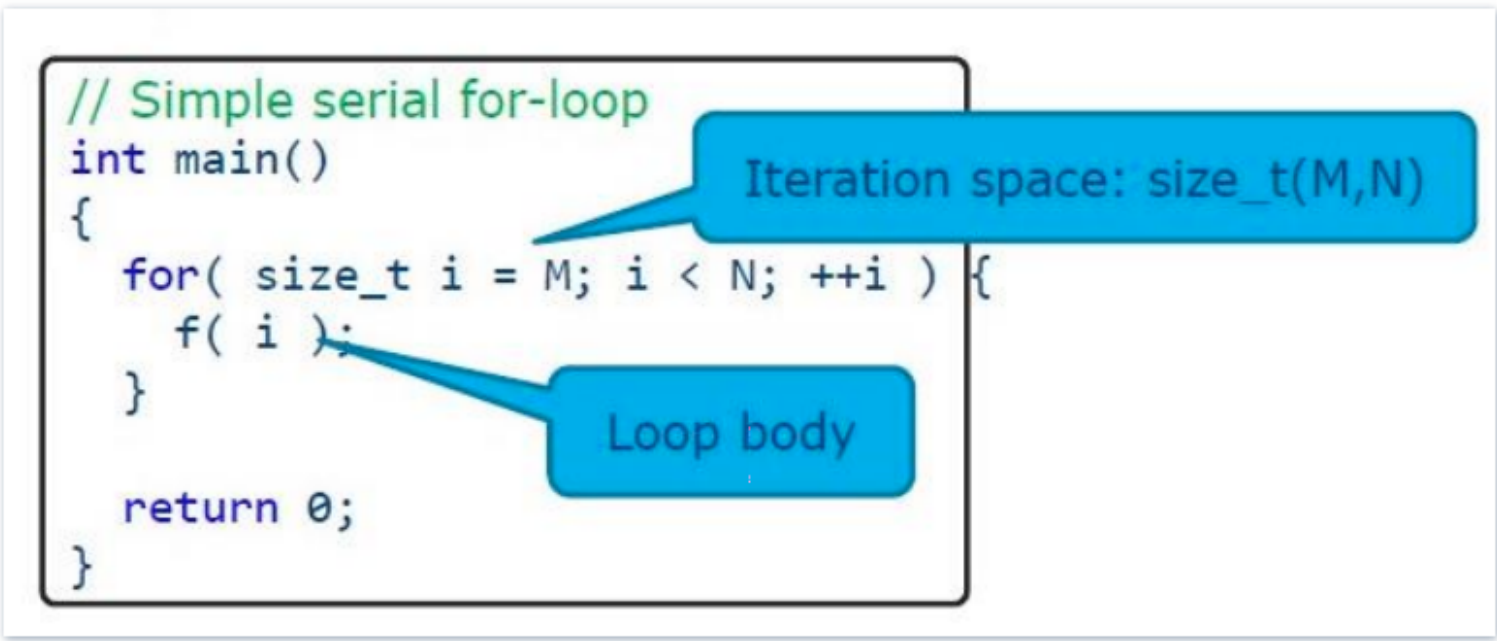


Aspectos da paralelização

- Algorítmico: Como dividir a computação em partes independentes que podem ser executadas em paralelo? Que tipos de recursos compartilhados são necessários? Quais os tipos de coordenação? Como minimizar os gastos gerais (redundância, coordenação, sincronização)?
- Scheduling/Mapping: Como as partes independentes da computação devem ser atribuídas aos processadores?
- Balanceamento de carga: Como as partes independentes podem ser atribuídas aos processadores, de modo que todos os recursos sejam utilizados de forma eficiente?
- Comunicação: Quando os processadores devem se comunicar? Como?
- Sincronização: Quando os processadores devem concordar/aguardar?

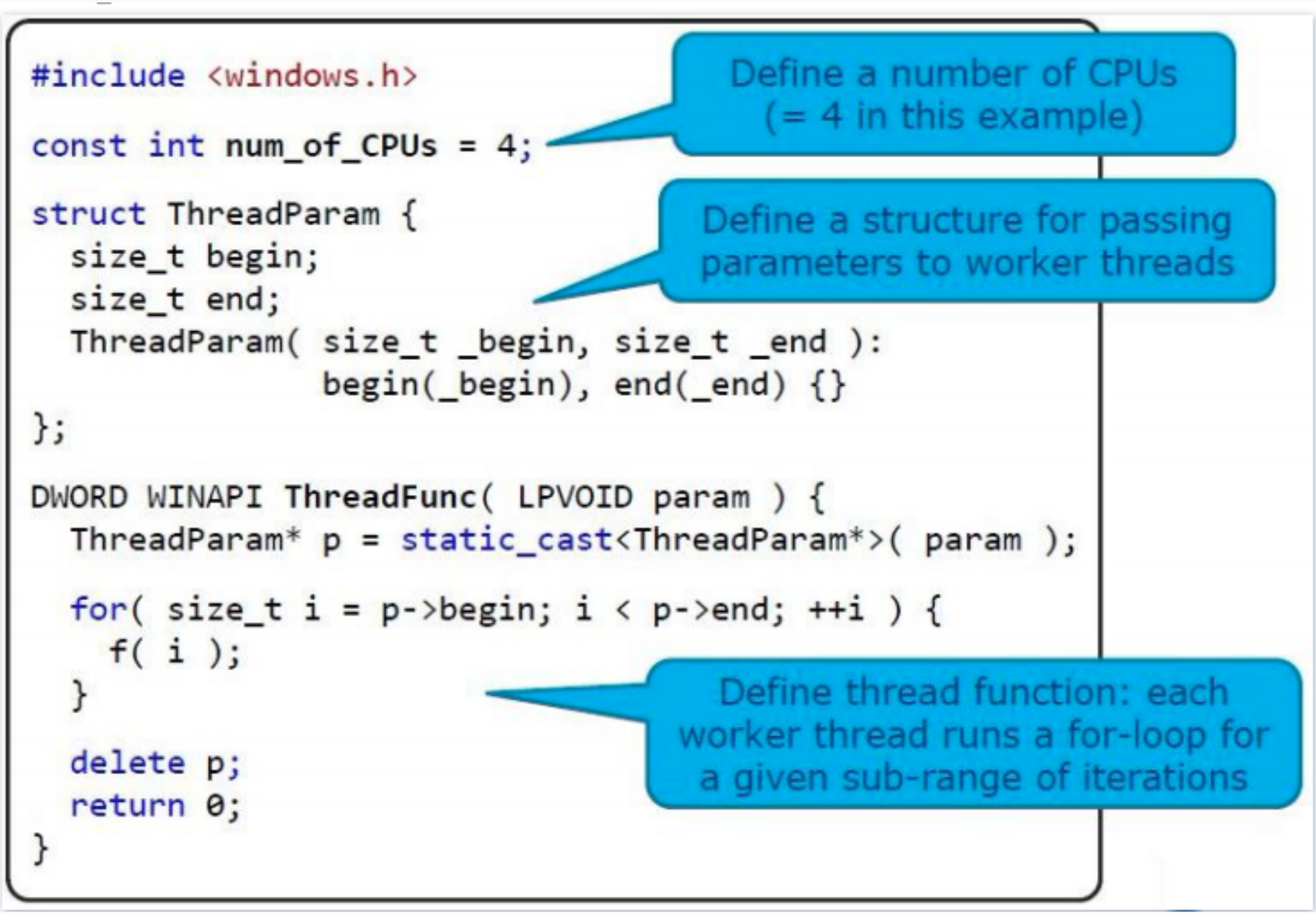
Um exemplo simples

- *Loop* é um exemplo simples de uma região de código que pode se beneficiar do paralelismo
- Vejamos uma das possíveis implementações de um "for-loop" paralelo



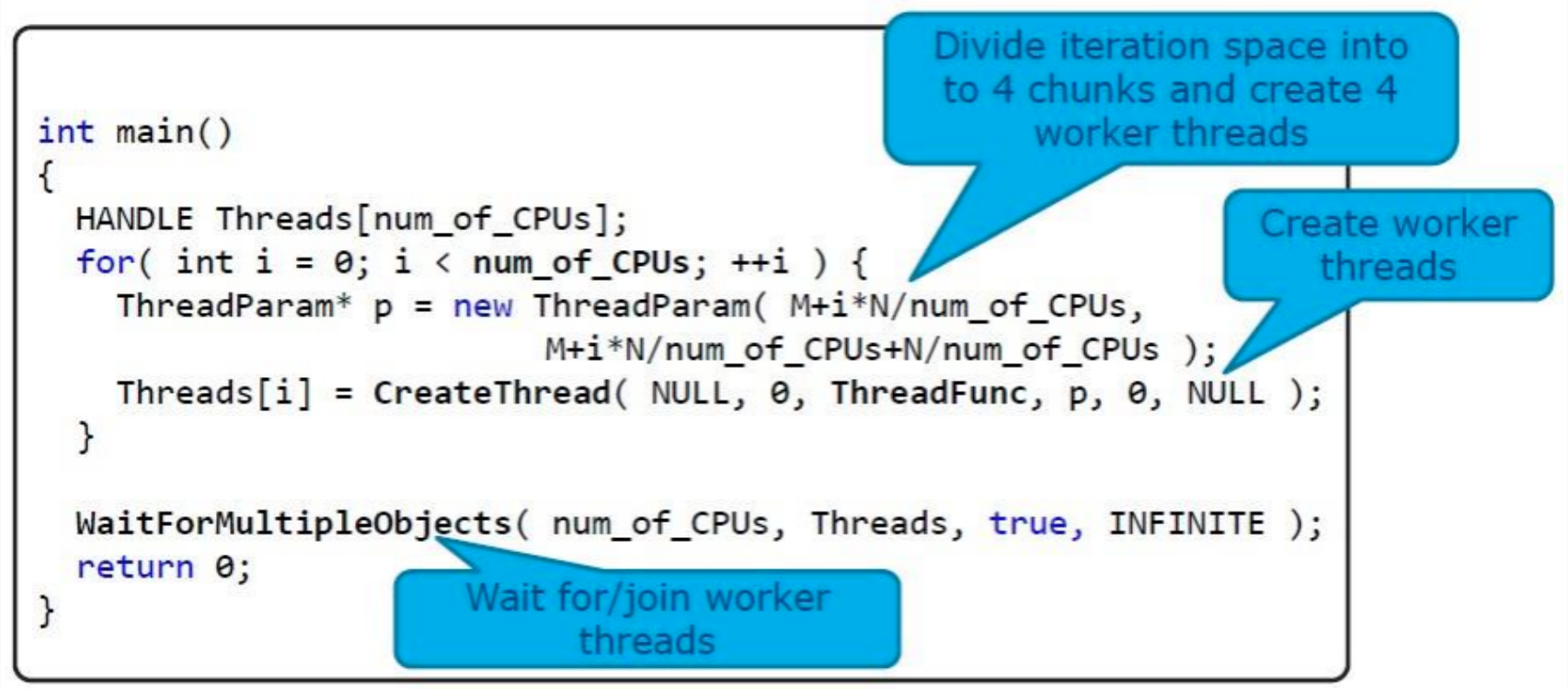
Coisas a considerar ao criar um “for-loop” paralelizado

Passo 1



Coisas a considerar ao criar um “for-loop” paralelizado

Passo 2

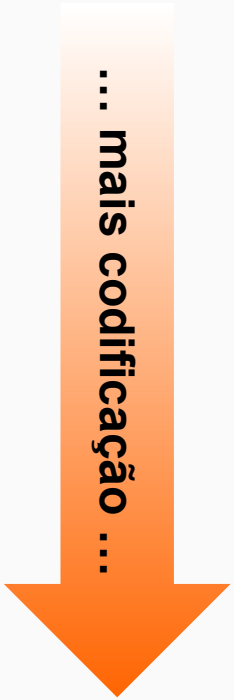


Muitas maneiras de melhorar uma implementação ingênua

Problemas com uma implementação ingênua	O que você pode fazer para melhorá-la
Trabalhar com número fixo de threads	Implementar uma função que determina o número ideal de threads
A implementação não é portável	Implementar funções com códigos específicos para cada SO suportado
Desempenho potencialmente fraco devido ao desbalanceamento da carga de trabalho dos processadores	Usar heurísticas para balancear a carga de trabalho entre as threads
A solução não é combinável	Bom... continue adicionando mais código... fazendo teste... e ajustes

Muitas maneiras de melhorar uma implementação ingênua

Problemas com uma implementação ingênua	O que você pode fazer para melhorá-la
Trabalhar com número fixo de threads	Implementar uma função que determina o número ideal de threads
A implementação não é portável	Implementar funções com códigos específicos para cada SO suportado
Desempenho potencialmente fraco devido ao desbalanceamento da carga de trabalho dos processadores	Usar heurísticas para balancear a carga de trabalho entre as threads
A solução não é combinável	Bom... continue adicionando mais código... fazendo teste... e ajustes



Desafios da programação paralela

- Algorítmico
  - Nem todos os problemas podem ser facilmente paralelizados
- Portabilidade
  - Suporte para a mesma linguagem/interface em diferentes arquiteturas
- Portabilidade de desempenho
- Paralelismo suficiente? (Lei de Amdahl)
- Balanceamento de carga
- Coordenação e Sincronização

Todas esses itens tornam a programação paralela ainda mais difícil do que a programação sequencial

Programação Paralela comparado à Programação Sequencial

- Tem diferentes custos, diferentes vantagens
- Requer diferentes (pouco familiar) algoritmos
- Deve-se usar diferentes abstrações
- Mais complexo de entender o comportamento de um programa
- Mais difícil controlar as interações dos componentes do programa
- Conhecimento/ferramentas/compreensão mais primitivo

Processamento paralelo

- Programas que acessam áreas compartilhadas de memória costumam ter regiões críticas
  - É necessário "coordenar" o acesso
- Com o advento de múltiplas unidades de processamento em um único computador, os programas tendem a ficar cada vez mais paralelizados e problemas de concorrência podem surgir
  - Esse problema ocorre com computadores com uma única unidade de processamento?

Exemplo - (falta) de sincronização

Pensando em eficiência

- **RAPIDEZ != EFICIÊNCIA**
- Apenas porque um programa é executado mais rápido em um computador paralelo, não significa que ele esteja usando o hardware de forma eficiente
  - Executar um programa 2x mais rápido em um computador com 10 processadores é um bom resultado?
- Perspectiva do programador: usar as capacidades da máquina fornecida de forma correta e eficiente

Dúvidas?