

O Modelo de Atores

MCZA020-13 - Programação Paralela

Emilio Franceschini

2018.Q3

Centro de Matemática, Computação e Cognição
Universidade Federal do ABC

A Lei de Moore

Programação Concorrente com Estado Compartilhado

O Modelo de Atores

Conclusão

Disclaimer

- Estes slides foram preparados para o curso de **Programação Paralela na UFABC**.
- Este material pode ser usado livremente desde que sejam mantidos, além deste aviso, os créditos aos autores e instituições.
- Estes slides contém alguns textos e figuras preparados por **Dror Bereznitsky**.

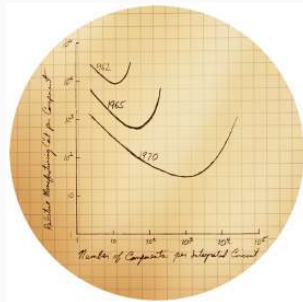
A Lei de Moore

É o fim da lei de Moore?

A Lei de Moore

O número de transistores em um chip vai dobrar aproximadamente a cada 18 meses.

Gordon E. Moore, 1965



É o fim da lei de Moore?

Andy Giveth, and Bill Taketh away

- Não importa o quão rápido os processadores se tornem, o software sempre encontra novas maneiras para consumir o ganho de desempenho.



É o fim da lei de Moore?

The Free Lunch

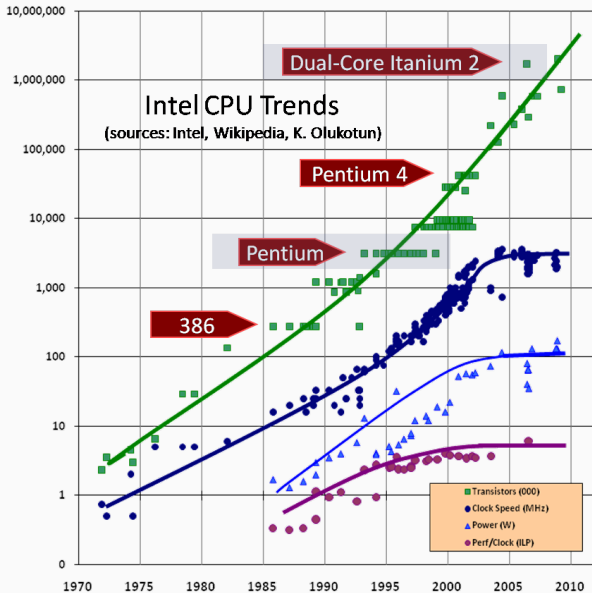
Ganhos de desempenho regulares, baratos e "gratuitos" sem nem mesmo ter a necessidade de lançar novas versões. Bastava esperar. . .

- Este tempo já passou.
- Ótimo artigo do Herb Sutter:

<http://www.gotw.ca/publications/concurrency-ddj.htm>



The Free Lunch is Over



The Free Lunch is Over

Mas por quê não temos processadores de 10GHz hoje em dia?

- No início dos anos 2000 a frequência de funcionamento dos processadores aumentou tanto que:
 - Os chips ficaram grandes o suficiente para a velocidade da luz ser uma barreira (os sinais não conseguem mais atravessar o chip todo em menos de um ciclo)
 - Frequências mais altas acabam causando problemas de dissipação de calor

The Free Lunch is Over

A era dos Multicores

- Os projetistas de hardware, então, se voltaram à produção de processadores multi-core
 - A incessante busca de desempenho passou então a ser uma tarefa compartilhada com os desenvolvedores de software
- Aumentam o desempenho total (paralelo) mas não mudam significativamente o desempenho sequencial
- De fato, o desempenho das CPUs tendem a ficar estáveis pelos próximos anos

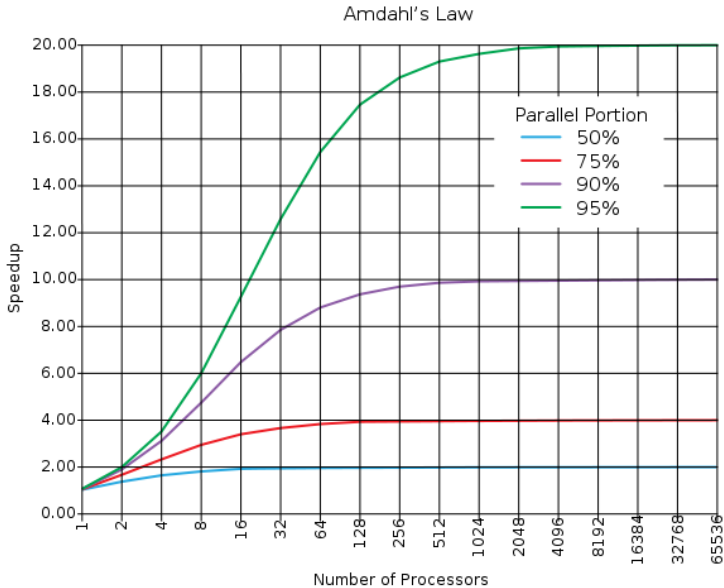
The Free Lunch is Over

A revolução da programação concorrente e paralela

- Acabou a mamata...
- Quer extrair até a última gota de desempenho do seu processador?

Você precisa desenvolver software **concorrente**

A Lei de Amdahl



Programação Concorrente com Estado Compartilhado

Concorrência com Estado Compartilhado

- Há um **compartilhamento de recursos** (variáveis/memória, dispositivos, ...)
- Estabelece-se um **mecanismo de controle e sincronização** de acesso a esses recursos
 - Travas (locks)
 - Semáforos
 - Monitores
 - ...

- Processos e Threads são a maneira do sistema operacional representar execuções concorrentes
- Threads, por definição, compartilham os recursos do processo ao qual pertencem com as demais threads do mesmo processo.
 - Exigem mecanismos de sincronização para acesso aos recursos compartilhados para garantir
 - Corretude
 - Consistência

Threads são utilizados pela maior parte das linguagens tradicionais

- C/C++
- Java
- Python
- C#
- Ruby
- ...

Threads não são legais!



Threads não são legais!

"Programas multi-threaded não triviais são incompreensíveis para um ser humano. . . "

Edward A. Lee, The Problem with Threads

- Diante da dificuldade de se utilizar threads corretamente, passou-se a procurar alternativas
 - Trocas de mensagens
 - MPI
 - Modelo de Atores
 - Memória transacional
 - Dataflow

O Modelo de Atores

O Modelo de Atores

Proposto originalmente por Hewitt *et al.* em 1973 e, mais tarde generalizado para concorrência em 1986 por Agha

- Baseado em princípios muito simples
 - Troca de mensagens assíncronas
 - Recepção seletiva de mensagens
 - Área de memória (*heap*) e laço de eventos privados
- Leve
 - Criado em quantidades excedendo o número de núcleos de processamento
 - **Desacopla** o número de atores do hardware



Aplicações baseadas em atores

- Têm a sua disposição ambientes de execução leves com distribuição automática e transparente de carga
- São altamente otimizados para máquinas com memória compartilhada
- São muito utilizados
 - **Linguagens:** Erlang, Scala, Akka, Kilim, Salsa, ...
 - **Applications:** WhatsApp, Facebook Chat, Chef Server, Twitter, CouchDB, ...
 - Atendem milhares de clientes simultaneamente em serviços dedicados executando em máquinas poderosas

Principais pontos sobre o Modelo de Atores

- Atores em vez de objetos
- Não há estado compartilhado entre atores
- Toda comunicação se dá por trocas de mensagens assíncronas
 - As mensagens são imutáveis
- Caixas postais podem fazer um buffer das mensagens recebidas
 - É o único canal de comunicação com um ator e age como uma fila com múltiplos produtores e um único consumidor



Atores podem...

- Reagir a alguma mensagem recebida executando um comportamento
 - Eles podem apenas alterar o próprio estado
 - ...ou mandar mensagens para outros atores.
- Modelo muito mais natural do que o modelo orientado a objetos!
 - Você é capaz de mudar o estado de algo na cabeça do seu colega? Isto é feito por compartilhamento de estado ou por troca de mensagens?
- Como os atores nunca compartilham o estado, nunca precisam competir por travas para acessar recursos compartilhados

Por debaixo dos panos. . .

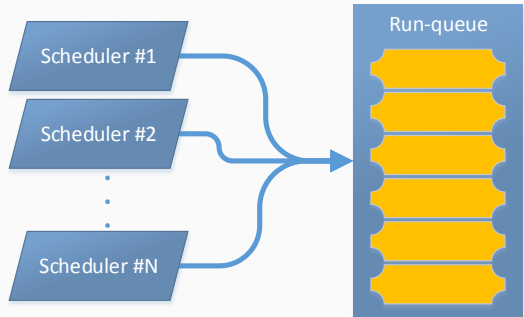
Os ambientes de execução do modelo de atores podem ser divididos em 2 principais categorias:

- Baseados em threads
 - Tem limitações sobre o número de atores e deixa a cargo do sistema operacional o escalonamento dos atores
- Baseados em eventos
 - Compostos de filas simples ou múltiplas de onde um pool de threads obtém as tarefas a serem executadas
 - Sistemas operacionais ainda não estão tão otimizados quanto poderiam para dar suporte a este tipo de aplicações

Single Run-Queue

Single Run-Queue

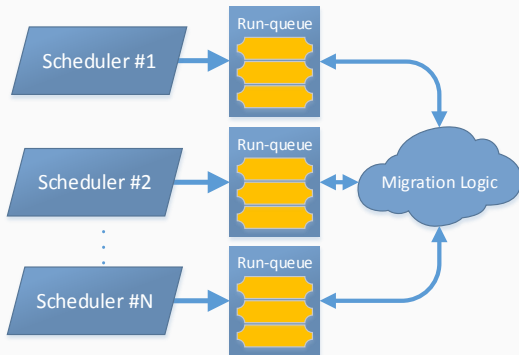
- Utilizado por Akka, Kilim, e VMs antigas de Erlang
- Pode se tornar um gargalo no desempenho
- Não garante soft-affinity



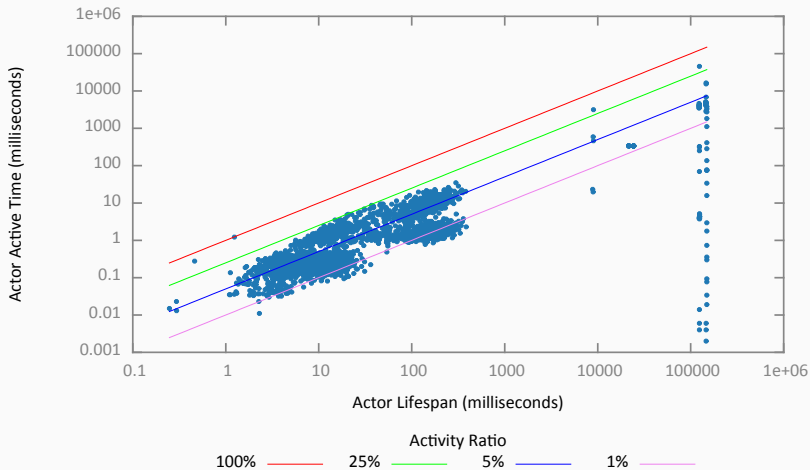
Multiple Run-Queue

Multiple Run-Queues

- Utilizado por Erlang, Kilim
- Colocação inicial/soft affinity
- Balanceamento de carga (baseado no tamanho das filas)
- *Work stealing*



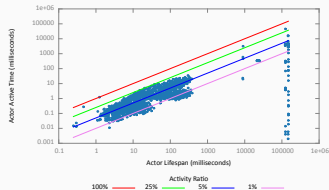
Entendendo o comportamento - Tempo de vida (1/2)



Couch DB

Entendendo o comportamento - Tempo de vida (1/2)

- Vivem pouco ($99.5\% < 1.5s$)
- Tempo de atividade médio 26%
- Atores com vida curta
 - Representam a maioria dos atores
 - Geralmente são trabalhadores
 - Maior atividade
 - Pela sua grande quantidade dividem o scheduler
- Atores com vida longa
 - Principalmente inativos
 - Mantêm o estado da aplicação
- O número de atores é independente do número de núcleos



Tamanho das mensagens

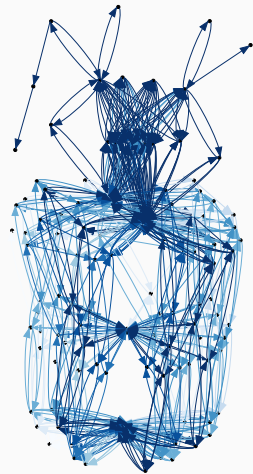
Analizando uma série de aplicações, percebemos que as mensagens são curtas

Application	Average Message Size
CouchDB	663 Bytes
Sim-Diasca	1088 Bytes
ErlangTW	960 Bytes
ehb	1024 Bytes
big	2112 Bytes
orbit_int	832 – 4608 Bytes

Grafo de comunicações

Alguns atores são mais populares que outros

- Natural já que algumas funções são mais requisitadas que outras
- Alguns são atores de coordenação, logo criam mais atores do que um ator comum
- Chamamos esses atores especiais **hubs**
- Os hubs normalmente só conversam com um conjunto pequeno de outros atores chamado de "grupo de afinidade".

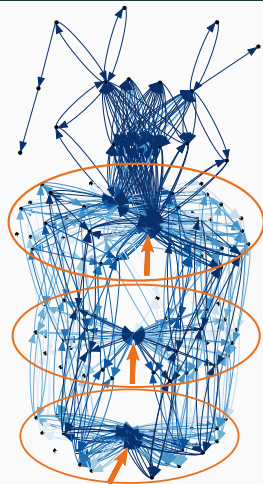


CouchDB
communication graph

Grafo de comunicações

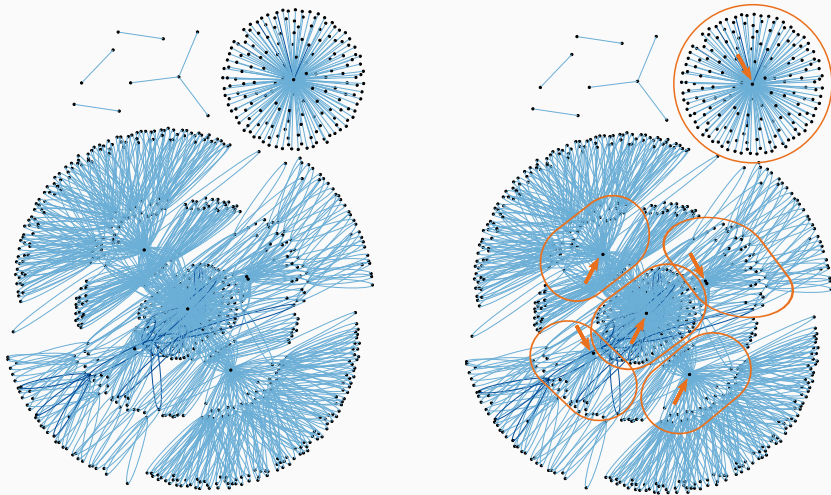
Alguns atores são mais populares que outros

- Natural já que algumas funções são mais requisitadas que outras
- Alguns são atores de coordenação, logo criam mais atores do que um ator comum
- Chamamos esses atores especiais **hubs**
- Os hubs normalmente só conversam com um conjunto pequeno de outros atores chamado de "grupo de afinidade".



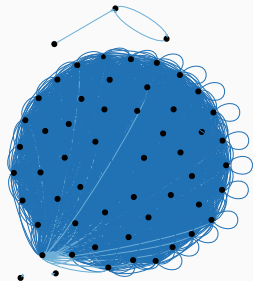
CouchDB
communication graph

Sim-Diasca - Grafo de Comunicação



O grafo de comunicação é específico por aplicação

- Neste caso é uma comunicação todos-com-todos
- Relativamente raro



- Ericson AXD 301 switch
 - Milhões de chamadas por minuto, 99,9999999% uptime
- Aplicação de chate do Facebook
- Servidores do Whatsapp
- RabbitMQ
 - AMQP de alto desempenho, 400.000 mensagens por segundo
- CouchDB
- Ejabberd XMPP server - jabber.org

Benefícios do modelo de atores

- Mais fácil de entender sistemas não triviais
- Nível de abstração alto
- Evita (ou facilita evitar)
 - Condições de corrida
 - Impasses
 - Starvation
 - Live locks
- Computação distribuída

Conclusão

- Acabou a mamata: se é preciso melhor desempenho, é preciso programar concorrentemente
- Aplicações precisam ser adaptadas. Quase nenhuma atualmente é preparada para isso.
- Programação baseada em estado compartilhado é difícil, propensa a erros, complicadíssima de se depurar, ...
- Existem modelos alternativos como
 - Concorrencia por passagem de mensagem
 - Memória transacional