Universidade Federal de São Carlos
Departamento de Computação

Sistemas Operacionais 2

2012/1

Hélio Crestana Guardia

http://www.dc.ufscar.br/~helio/so2

Objetivos

- Os alunos deverão conhecer, através de estudo de casos, questões relacionadas à **programação** em Sistemas Operacionais. Abordar o acesso aos **serviços** oferecidos em sistemas compartilhados e com múltiplos processadores e estudar suas **implementações**.
- Assim, enquanto a disciplina Sistemas Operacionais 1 trata dos aspectos da implementação dos mecanismos de gerenciamento dos recursos disponíveis, **Sistemas Operacionais 2** concentra-se em:
  - Estudar **como** o SO implementa suas políticas
  - Estudar **quais** recursos são oferecidos pelo SO para os programas
  - Conhecer técnicas para criação de programas que usam de maneira **eficiente** os recursos do SO, tanto em ambientes com memória compartilhada quanto naqueles com comunicação por passagem de mensagem.

Ementa

- Chamadas de sistema
- Serviços de entrada e saída
- Gerenciamento de processos e *threads*
- Comunicação e sincronização com memória compartilhada (IPC)
- Programação distribuída: passagem de mensagem, sincronização e execução remota de código
- Programação paralela: tarefas, comunicação e sincronização
- Algoritmos de programação paralela

Bibliografia

Conceitos

- Deitel,H.M.; Deitel,P.J. and Choffnes,D.R. *Sistemas Operacionais*. 3a edição. Pearson, 2005.
- Tanembaum, A. S. *Sistemas Operacionais: Projeto e Implementação*. 3a edição. Bookman, 2008.
- Andrews, G. R. *Foundations of Multithreaded, Parallel, and Distributed Programming*. Addison-Wesley, 2000.
- Tanembaum, A. S. *Sistemas Operacionais Modernos*. 3a edição, Pearson Pentice Hall, 2010.
- Wilkinson, B. and Allen, M. *Parallel Programming: Techniques and Applications Using Networked Workdstations and Parallel Computers*. Pearson Prentice Hall, 2005.
- Foster, I. *Designing and Building Parallel Programs*. MIT Press 1999. www-unix.mcs.anl.gov/dbpp.
- Bach, M. J. *The Design of the Unix Operating System*. Prentice-Hall, 1986.

Programação

- Mitchell, M.; Oldham, J.; and Samuel *Advanced Linux Programming* New Riders Publishing, 2001.
- Kernighan, B.W. *The Unix Programming Environment*. Prentice-Hall, 1984.
- Rochkind, M.J. *Advanced Unix Programming*. Prentice-Hall, 1985.
- Stevens, W. R. *Unix Network Programming: Interprocess Communications*. 2nd ed. Prentice Hall, 1999.
- Stevens, W. R. *Unix Network Programming: Networking APIs: Sockets and XTI*, 2nd ed. Prentice Hall, 1999.
- Snir, M. et. al. *MPI - The Complete Reference. Vol.1 The MPI Core*. MIT, 1998, Second Edition.
- Gropp, W. e. al. MPI - *The Complete Reference. Vol.2 The MPI Extensions*. MIT, 1998, Second Edition.
- Robbins, K. A. and Robbins, S. *Practical Unix Programming: A Guide to Concurrency, Communication, and Multithreading*. Prentice-Hall, 1996.

Sistema Operacional

Funções:

- Gerente de recursos
- Máquina virtual mais fácil de usar e programar
- Interface entre os usuários, e seus programas, e os recursos disponíveis

Aspectos de interesse:

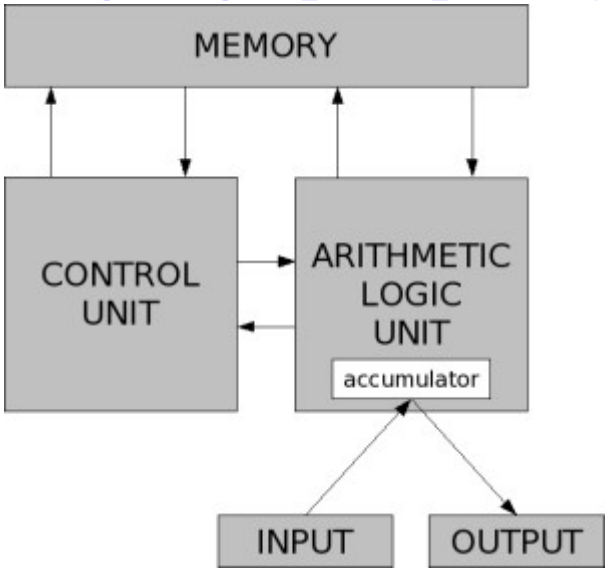
- *Hardware* (processadores e elementos funcionais)
- Estratégias de implementação (arquitetura)
- Políticas e serviços oferecidos (*syscalls*)

SO: *hardware* = recursos

Arquitetura de Von Neumann:

- Programa = sequência **de instruções** na memória
- Registrador (**PC**) indica palavra de memória que contém próxima instrução a executar
- Decodificação e execução das instruções feitas a partir de um registrador interno da CPU (**IR)**

http://commons.wikimedia.org/wiki/Image:Von_Neumann_architecture.png



Operação:

```
Loop {
    IR = Mem[PC]
    PC = PC +1
    Executa IR
    Testa e trata INT
}
```

 # Busca instrução; uso do cache e mecanismos de prefetching, branch prediction, ...
 # Incrementa ponteiro de instruções. Incremento equivalente ao tamanho da palavra...
 # Decodifica e executa instrução. Lógica no nível de micro-programa.
 # Verifica interrupção: assíncrona (externa), trap, instrução (int)

SO: *hardware* = recursos

- Controladores de dispositivos (canais) com capacidade de operação independente
- Interconexão de controladores de dispositivos (barramentos):
  - Dados
  - Endereçamento
  - Controle
- Leitura e escrita entre dispositivos e memória:
  - Passando pela CPU (instruções *in* e *out*)
  - Usando DMA
- Detecção de alterações de estado dos controladores:
  - *Polling* (consulta pelo SO via barramento
  - Notificações assíncronas via **interrupções**
- **Interrupções**:
  - Externas
  - Traps
  - Instrução (int)

SO: Estratégias

- Entidades "escalonáveis": **processos** e ***threads***
  - Representação de processos
  - Estados dos processos
  - Trocas de contexto
- Compartilhamento do processador:
  - **Multiprogramação**: sobreposição de Entrada e Saída (disco e rede) com execução de instruções
  - **Fatias de tempo**: interrupções periódicas
- **Concorrência**: ambientes monoprocessados
- **Paralelismo**: **SMP**, ***Multicore***, **HTT (***Hyper Threading Technology***)**, ou outras arquiteturas multiprocessadas.

SO: Serviços (*System Calls*)

- Todo **sistema operacional** provê uma **interface** para que programas solicitem **serviços** do núcleo.
- Variações de Unix oferecem funções bem definidas, chamadas ***system calls***, ou **chamadas de sistema**.
- Definição das chamadas feita em **C**, independentemente do mecanismo utilizado para solicitação dos serviços.
- Cada chamada de sistema normalmente tem uma função com o mesmo nome definida em **C**.
- Processo de usuário chama função usando a sequência de chamada normal da linguagem.
- Função faz chamada apropriada ao serviço do *kernel,* colocando valores nos **registradores do *hardware*** (e possivelmente também na **pilha**) e chamando instrução de **interrupção**.

SO: Serviços (*System Calls*)

- *_llseek(2), _newselect(2), _sysctl(2), **accept(2)**, access(2), acct(2), adjtimex(2), afs_syscall, alarm(2), bdflush(2), **bind(2)**, break, brk(2), cacheflush(2), capget(2), capset(2), chdir(2), **chmod(2)**, chown(2), chown32, chroot(2), clone(2), close(2), **connect(2)**, creat(2), create_module(2), delete_module(2), dup(2), dup2(2), **execve(2)**, exit(2), fchdir(2), fchmod(2), fchown(2), fchown32, fcntl(2), fcntl64, fdata-sync(2), flock(2), **fork(2)**, fstat(2), fstat64, fstatfs(2), fsync(2), ftime, ftruncate(2), ftruncate64, get_kernel_syms(2), getcwd(2), getdents(2), getdents64, getegid(2), getegid32, geteuid(2), geteuid32, getgid(2), getgid32, getgroups(2), getgroups32, getitimer(2), getpagesize(2), getpeername(2), getpmsg, getpgid(2), getpgrp(2), **getpid(2)**, getppid(2), getpriority(2), getresgid(2), getresgid32, getresuid(2), getresuid32, **getrlimit(2)**, **getrusage(2)**, getsid(2), getsockname(2), getsockopt(2), gettid, gettimeofday(2), getuid(2), getuid32, gtty, idle, init_module(2), ioctl(2), ioperm(2), iopl(2), ipc(2), **kill(2)**, lchown(2), lchown32, link(2), listen(2), lock, lseek(2), lstat(2), lstat64, madvise(2), mincore(2), mkdir(2), mknod(2), mlock(2), mlockall(2), **mmap(2)**, modify_ldt(2), mount(2), mprotect(2), mpx, mremap(2), msync(2), munlock(2), munlockall(2), munmap(2), **nanosleep(2)**, nfsservctl(2), **nice(2)**, oldfstat, oldlstat, oldolduname, oldstat, oldumount, olduname, open(2), pause(2), personality(2), phys, **pipe(2)**, pivot_root(2), poll(2), prctl(2), pread(2), prof, profil, ptrace(2), putpmsg, pwrite(2), query_module(2), quotactl(2), **read(2)**, readahead, readdir(2), readlink(2), readv(2), reboot(2), recv(2), **recvfrom(2)**, recvmsg(2), rename(2), rmdir(2), rt_sigaction, rt_sigpending, rt_sigprocmask, rt_sigqueueinfo, rt_sigreturn, rt_sigsuspend, rt_sigtimedwait, sched_get_priority_max(2), sched_get_priority_min(2), sched_getparam(2), sched_getscheduler(2), sched_rr_get_interval(2), sched_setparam(2), **sched_setscheduler**(2), sched_yield(2), security, select(2), sendfile(2), send(2), sendmsg(2), **sendto(2)**, setdomainname(2), setfsgid(2), setfsgid32, setfsuid(2), setfsuid32, setgid(2), setgid32, setgroups(2), setgroups32, sethostname(2), setitimer(2), setpgid(2), setpriority(2), setregid(2), setregid32, setresgid(2), setresgid32, setresuid(2), setresuid32, setreuid(2), setreuid32, setrlimit(2), setsid(2), setsockopt(2), settimeofday(2), setuid(2), setuid32, setup(2), sgetmask(2), shutdown(2), sigaction(2), sigaltstack(2), signal(2), sigpending(2), sigprocmask(2), sigreturn(2), sigsuspend(2), socket(2), socketcall(2), socketpair(2), ssetmask(2), stat(2), stat64, statfs(2), stime(2), stty, swapoff(2), swapon(2), symlink(2), sync(2), sysfs(2), sysinfo(2), syslog(2), time(2), times(2), truncate(2), truncate64, ulimit, umask(2), umount(2), uname(2), unlink(2), uselib(2), ustat(2), utime(2), vfork(2), vhangup(2), vm86(2), vm86old, wait4(2), **waitpid(2)**, **write(2)**, writev(2).*

Como usar os recursos computacionais de maneira eficiente?

- Exploração (via serviços do SO) dos múltiplos **elementos** funcionais **autônomos** do hardware (canais de E/S e processadores) implica organizar o *software* a ser executado para conter **partes** relativamente **independentes**.

- Modelos de aplicações que exploram o paralelismo:
  - Aplicações *multi-threaded*
  - Aplicações distribuídas
  - Aplicações paralelas

Modelos de aplicação

- ***Software multithread*** gerencia múltiplas atividades independentes, tais como sistemas de janelas em computadores pessoais ou *workstations,* sistemas de tempo-real que controlam ambientes externos, etc.
- Exemplos de **computação distribuída** incluem sistemas de arquivo em um sistema em rede, bancos de dados para sistemas bancários, venda de passagens aéreas, servidores web na Internet, etc.
- Exemplos de **computação paralela** incluem computação científica que modela e simula fenômenos climáticos, bioinformática, como na análise do efeito de novas drogas, processamento gráfico e de imagens, incluindo a criação de efeitos especiais em filmes, grandes problemas combinatoriais ou de otimização, etc.

O grau de **dependência** e a forma de **interconexão** dos processadores são importantes no suporte ao **modelo de programação** selecionado.

Modelos de programação

Programação com Memória Compartilhada:

- *Threads*: programas são decompostos em sequências paralelas (*threads*), que compartilham variáveis dentro do escopo do programa.
  - ***Pthreads***: biblioteca padronizada para manipulação de *threads*
- Programas escritos em linguagem de programação sequencial, incluindo diretivas de compilação para declarar variáveis compartilhadas e especificar paralelismos.
  - **OpenMP**: biblioteca para paralelização automatizada de código Fortran e C(++). Usando diretivas inseridas no código, tornou-se um padrão para sistemas com memória compartilhada.

Modelos de programação

Programação com Memória Distribuída (*clusters*):

- Computação paralela baseada na **passagem de mensagem** (*Message Passing Parallel Programming*)
  - ***Message-Passing Interface*** (MPI): padrão

Computação em Grade (*Grid Computing*):

- Computação paralela baseada na execução de código remotamente
  - Ativação remota de código
  - *Web/Grid Services*
- Implementação padronizada: **Globus** (OGSI)

Computação em nuvem (*Cloud Computing*):

- Computação distribuída através de recursos virtualizados, acessíveis via Internet, na forma de **serviços**
- Usuários não têm conhecimento da tecnologia que provê os serviços
- Conceitos: *Infrastructure as a service (IaaS), Platform as a service (PaaS) e Software as a service (SaaS)*

Por quê computação paralela?

Demanda por poder computacional é sempre crescente:

- Desejo de resolver **novos** problemas
- Necessidade de resolver problemas computacionais já tratados:
  - com **melhor precisão**
  - de maneira **mais rápida**
- Limitação na capacidade de integração de circuitos (tamanho, consumo de energia, dissipação de calor, …) e de aumento de desempenho com um único chip

Problemas computacionais

Aplicações atuais em muitos casos envolvem a manipulação de **grandes volumes de dados**, o que requer processamento extensivo.

Exemplos:

- Bancos de dados paralelos e *data mining*
- Exploração mineral
- Máquinas de busca na Web
- Serviços baseados na Web
- Suporte para diagnósticos auxiliados por computador
- Gerenciamento de grandes empresas
- Computação gráfica e realidade virtual
- Suporte para tecnologias multimídia
- Ambientes para trabalho cooperativo

Grand Challenge Problems

"A grand challenge problem is one that cannot be solved in a reasonable amount of time with today's computers."

Exemplos (*http://en.wikipedia.org/wiki/Grand_Challenge_problem*):

- *Applied Fluid Dynamics*
- *Meso- to Macro-Scale Environmental Modeling*
- *Ecosystem Simulations*
- *Biomedical Imaging and Biomechanics*
- *Molecular Biology*
- *Molecular Design and Process Optimization*
- *Cognition*
- *Fundamental Computational Sciences*
- *Grand-Challenge-Scale Applications*
- *Nuclear power and weapons simulations*

Programação Paralela

Problemas computacionais complexos normalmente apresentam características que favorecem o uso da computação paralela:

- Podem ser **divididos** em partes distintas que podem ser executadas simultaneamente
- Podem ter **diversas instruções** sendo executadas ao mesmo tempo
- São executados em **menor tempo** quando **múltiplos** recursos computacionais são utilizados

Programação Paralela

- Programação paralela trata da programação de **múltiplos computadores**, ou de computadores com **múltiplos processadores**, para resolver um problema mais rapidamente do que é possível com um único processador.
- **Ideia**: resolução de um problema normalmente pode ser dividida em tarefas menores, que podem ser executadas simultaneamente com alguma coordenação.
- Motivos:
  - Tratar problemas maiores, que requerem mais **processamento** ou mais **memória** que a normalmente disponível em um único sistema.
  - Prover tolerância a falhas, ...
- **Problema**: ***N*** computadores trabalhando de maneira simultânea podem obter um resultado ***N*** vezes mais depressa?

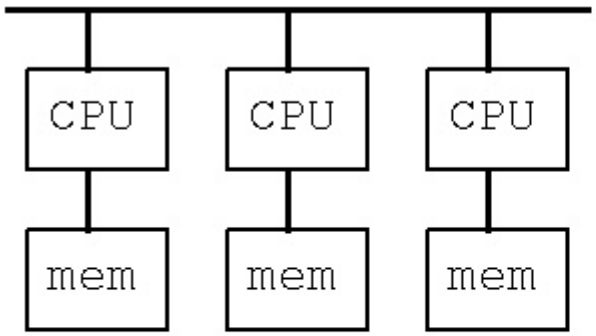
Programação Paralela: etapas

- **Decomposição** do programa ou dos dados
- **Distribuição** dos programas ou dados
- **Coordenação** do processamento e das comunicações

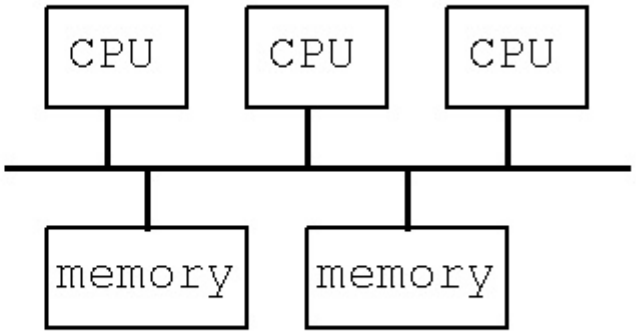
- Aspectos:
  - Arquitetura paralela
  - Formas de comunicação

Tipos de Computadores Paralelos

- Multi**processadores** com memória compartilhada
- Multi**computadores** com memória distribuída



Distributed Memory Multiprocessor



Shared Memory Multiprocessor

Taxonomia de Flynn (*)

Classificação de computadores de alto desempenho, baseada na forma de manipulação de instruções e dados:

- **SISD**: *Single instruction, single data stream*
- **MISD**: *Multiple instruction, single data stream*
- **SIMD**: *Single instruction, multiple data streams*
- **MIMD**: *Multiple instruction, multiple data streams*

(*) M.J. Flynn, *Some computer organizations and their effectiveness*, IEEE Transactions on Computing, C-21, (1972) 948-960.

Taxonomia de Flynn

- ***Single instruction, single data stream (SISD)***: computador convencional com **uma** CPU, sem paralelismo de instruções ou dados, como um PC ou um *mainframe*.
- ***Multiple instruction, single data stream (MISD)***: computador **hipotético**, em que múltiplos processadores atuariam sobre um único fluxo de dados. Pode ser empregado em casos de paralelismo para redundância.

Taxonomia de Flynn

- ***Single instruction, multiple data streams (SIMD)***: computador que explora múltiplos fluxos de dados com um único fluxo de instruções. Normalmente possui grande número de processadores, que executam a mesma operação de maneira coordenada (*lock-step*) sobre dados diferentes, como um *array processor ou vectorprocessor*.

Uma variação dessa classificação inclui:

- ***Single Program, multiple data streams (SPMD)***: múltiplos processadores autônomos executando simultaneamente o mesmo conjunto de instruções, de maneira independente, sobre dados distintos.

Taxonomia de Flynn

- ***Multiple instruction, multiple data streams (MIMD)***: consistem de múltiplos processadores autônomos, que executam diferentes instruções sobre diferentes conjuntos de dados. Exemplos dessa arquitetura incluem os sistemas distribuídos.

Uma sub-divisão desses sistemas pode ser feita em função do compartilhamento de memória:

- **Sistemas com memória compartilhada**: apresentam múltiplas CPUs que compartilham o mesmo espaço de endereçamento, como os sistemas SMP.
- **Sistemas com memória distribuída**: neste caso, cada CPU possui sua própria memória. Redes de comunicação podem permitir trocas de dados, usando diferentes topologias e tecnologias de comunicação, normalmente transparentes para as aplicações.

Aspectos do paralelismo dos processadores

- Acoplamento dos processadores
- Simetria de multiprocessamento
- *Multi-Core* e *Hyper-Threading*
- Hierarquias de memória

Acoplamento dos Processadores

Processor coupling

- Sistemas **fortemente acoplados** (*tightly-coupled*):
  - Múltiplas CPUs conectadas no nível do barramento interno (*bus*)
  - Acesso a **memória compartilhada** (SMP) ou hierarquia de memória com acesso não uniforme (NUMA).
  - Sistemas ***multi-core*** são exemplo extremo de acoplamento.
- Sistemas **fracamente acoplados** (*loosely-coupled*):
  - *Clusters*, tipicamente, compostos de nós simples ou duais, interligados por rede de comunicação rápida.
  - Mais baratos para serem integrados (agrupados) e atualizados

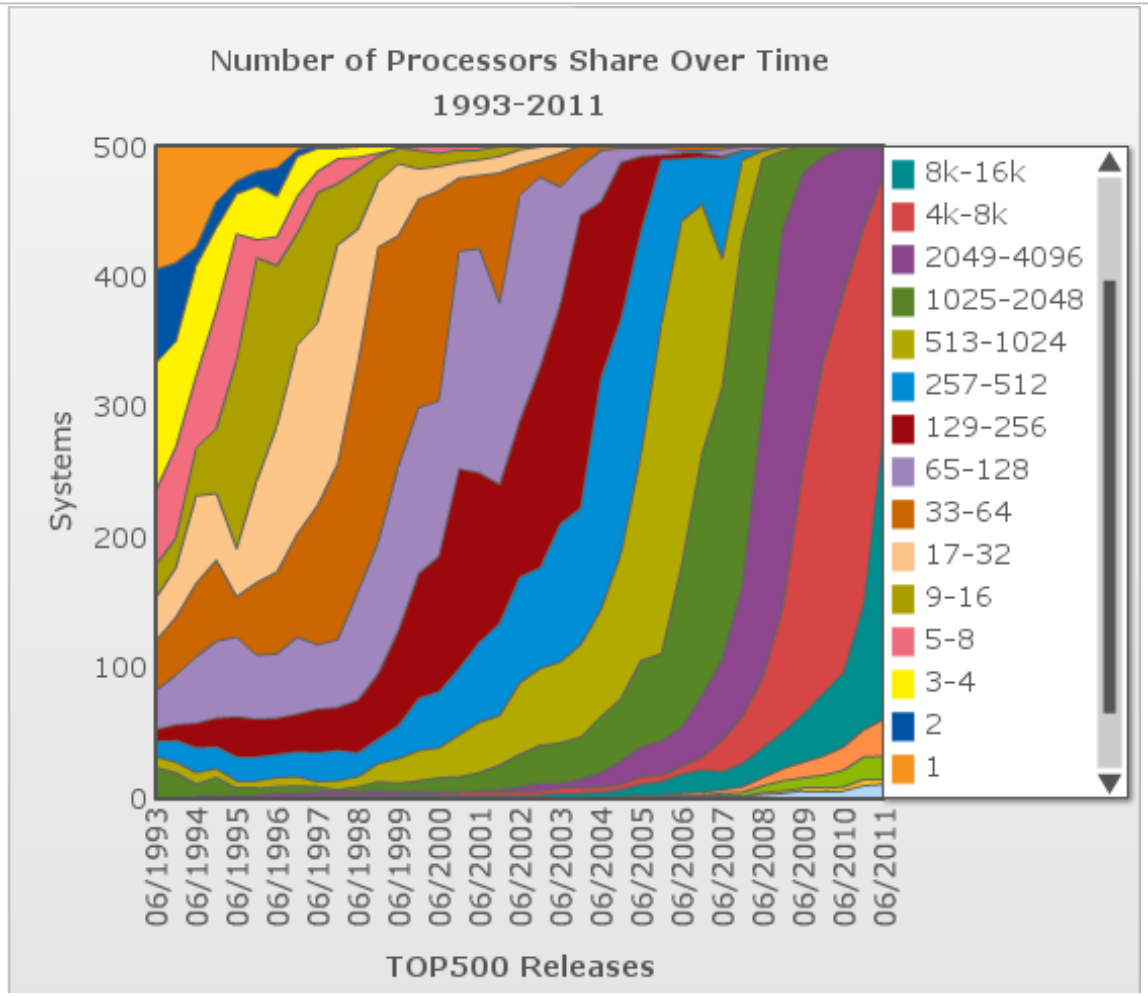
Simetria de Multiprocessamento

- Em um sistema multiprocessado, CPUs podem ter as mesmas funcionalidades ou pode haver funções executadas por apenas algumas delas.
- Questões de ***hardware*** e do **Sistema Operacional** determinam a **simetria** entre os processadores.
- Tratamento de **interrupções** e a execução de **serviços em *kernel mode*** são exemplos de atividades que podem não ser balanceadas entre todos os processadores disponíveis.
- Simplicidade do modelo assimétrico pode gerar gargalos no desempenho.
- Sistemas em que todas as CPUs são simétricas são chamados ***Symetric Multiprocessing* (SMP)**
- Em *clusters*, o processamento é tipicamente assimétrico

Computadores: Limites e tendências

- **Limitações físicas** dificultam criar computadores mais rápidos:
  - **Transmissão de dados**: velocidade do computador depende da taxa de transmissão no *hardware*. Luz: 30 cm/ns; transmissão em cobre: 9 cm/ns. Aumento da velocidade implica diminuir distância entre elementos.
  - **Miniaturização**: tecnologia de processadores tem permitido aumentar número de transistores em ***chip***, mas há limitações sobre quão pequenos componentes podem ser.
  - **Economia**: custo para desenvolver processador mais rápido é cada vez maior.
- **Vantagens** do paralelismo:
  - Processadores **comerciais** oferecem desempenho cada vez melhor, incluindo múltiplos processadores no mesmo ***chip*** (*dual-core, multi-core*) e suporte para execução simultânea de várias atividades (*Hiper-Threading*).
  - **Redes rápidas**: tecnologias de rede oferecem interligação da ordem de 1 e até 10 Gbps em preços acessíveis para grupos de máquinas.
- Uso de **N** processadores comuns **interligados** é mais **barato** que 1 processador **N** vezes mais rápido com **mesmo desempenho**.

Alto desempenho: tendências



http://www.top500.org/lists/2011/06

Hyper-Threading

- ***Hyper-threading*** (***Hyper-Threading Technology – HTT***) é uma tecnologia **Intel** para a execução simultânea de tarefas na arquitetura **Pentium 4**, aprimorando o suporte de *threading* que surgiu nos processadores **Xeon**.
- Desempenho do processador é melhorado fazendo com que ele execute processamento útil quando estaria ocioso, como nas falhas de dados em cache (***cache miss***), ***branch misprediction***, ou **dependência de dados**.
- Operação ocorre duplicando as partes específicas do processor relacionadas com a manutenção do ***estado do hardware***.

http://www.intel.com/technology/hyperthread

Multi-Core

- Processadores multi-core combinam 2 ou mais processadores independentes em um mesmo Circuito Integrado
- Sistema operacional “enxerga” cada um dos *cores* com um processador distinto, com seus próprios recursos de execução
- Paralelismo no nível de tarefas é facilitado

http://www.intel.com/software/multicore

NUMA: Non-Uniform Memory Access / Architecture

- Arquiteturas NUMA são uma extensão dos sistemas SMP
- ***Non-Uniform Memory Access*** trata do projeto de acesso à memória em sistemas multiprocessados, utilizando módulos de memória para cada processador
- Técnicas empregadas incluem o uso de *caches* internos ao processador cada vez maiores e algoritmos sofisticados para evitar *cache miss*
- Presença dos dados sendo manipulados nos caches locais favorece o paralelismo nos acessos
- *Hardware* (ou *software*) adicional provê a cópia automática dos dados entre os módulos de memória dos processadores, normalmente por demanda, quando os dados são compartilhados
- A manutenção da consistência dos dados compartilhados nos diversos caches é um problema significativo e um gargalo no desempenho desses sistemas.
- Arquiteturas ***Cache-coherent NUMA*** (**ccNUMA**) usam *hardware* especial para manter a consistência dos dados compartilhadosnos *caches*.
- Coerência dos dados é obtida com trocas de informações entre controladores de cache nos processadores quando um mesmo bloco de dados é mantido em vários caches.

Aspectos da programação paralela e concorrente

Granularidade

- É definida pela razão entre computação e comunicação
- **Granularidade Fina** (*fine-grain*) ou paralelismo com alto grau de acoplamento (*tightly coupled parallelism*):
  - Tarefas são relativamente pequenas em termos de código e tempo de execução.
  - Neste caso, há pouca transferência de dados entre as tarefas.
- **Granularidade Grossa** (*coarse grain*) ou paralelismo com pequeno grau de acoplamento (*loosely coupled*):
  - Tarefas relativamente grandes,
  - Comunicação freqüente entre as tarefas
- Potencial de paralelização é maior com granularidade fina mas, nesse caso há mais sobrecargas de comunicação e sincronização.

Aspectos da programação paralela e concorrente

Atributos desejáveis de algoritmos e programas paralelos:

- **Concorrência** (*concurrency*): capacidade de executar diversas ações simutaneamente.
- **Escalabilidade** (*scalability*): capacidade de se adaptar a ambientes com maiores números de processadores (escalabilidade de arquitetura).
- **Localidade** (*locality*): taxa alta de acessos a dados locais em relação a acessos remotos.
- **Modularidade** (*modularity*): possibilidade de decomposição em componentes mais simples, desenvolvidos separadamente como módulos independentes, combinados para a realização do programa. Usando interfaces definidas, modificações dos módulos são facilitadas. Projetos modulares reduzem a complexidade dos programas e facilitam a reutilização de código.

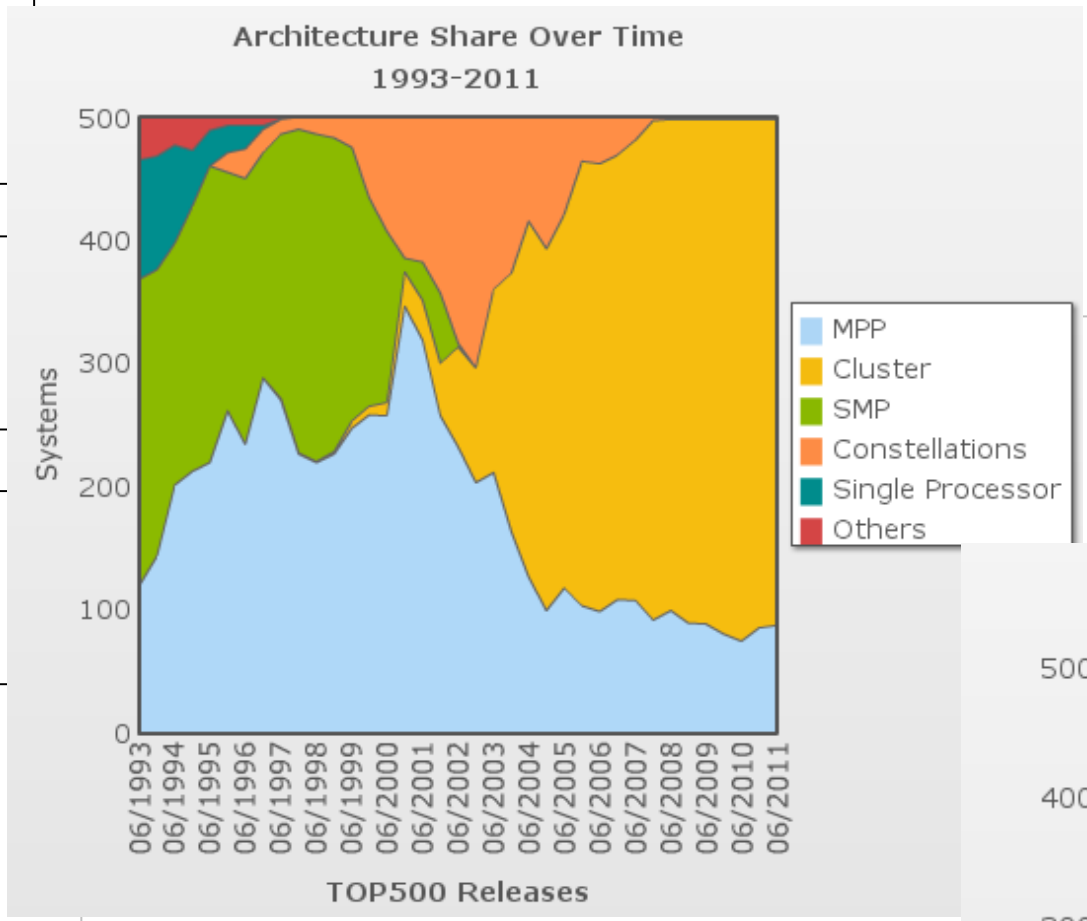
Aspectos da programação paralela e concorrente

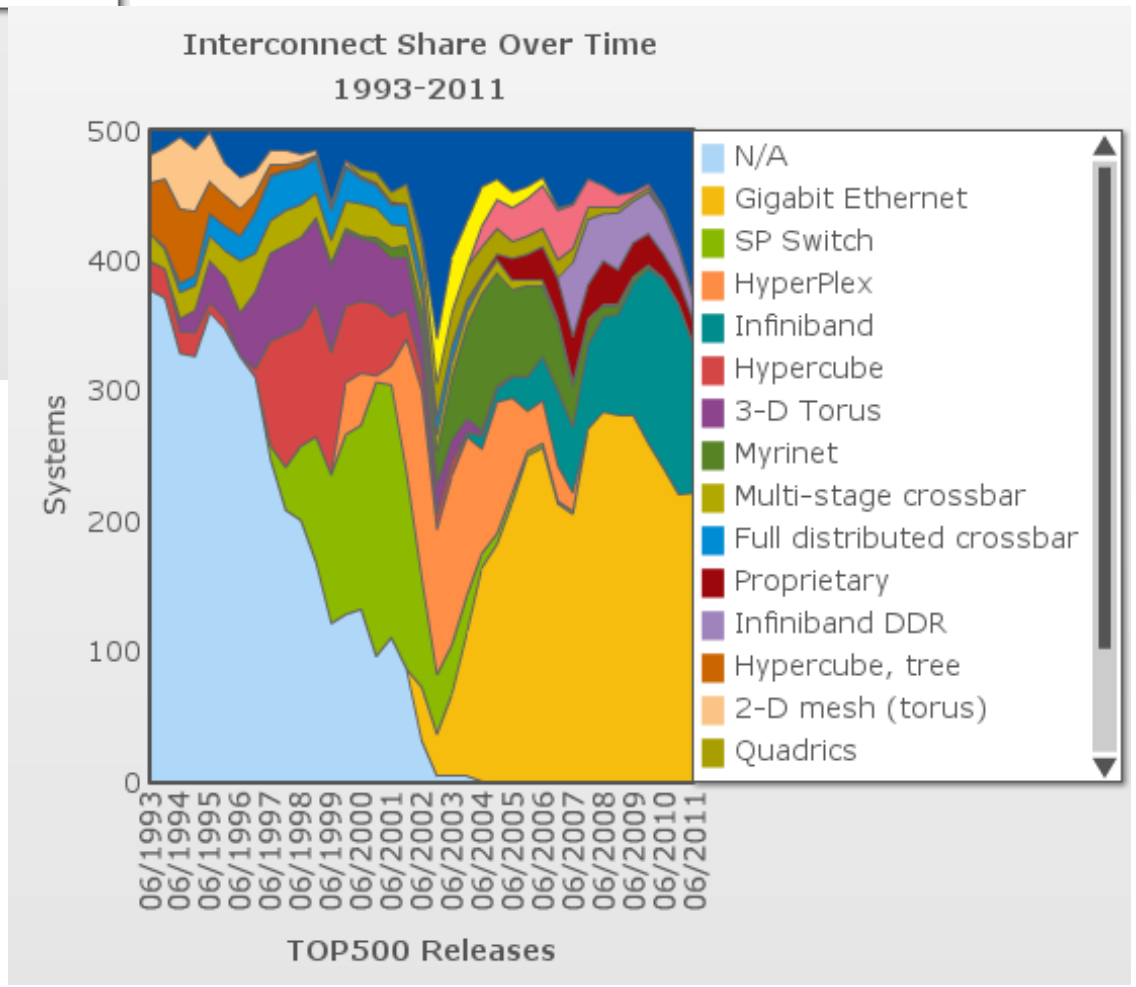
Localidade

- Acessos à memória local são menos custosos que acessos remotos.
- Operações ***read*** e ***write*** são menos onerosas que ***send*** e ***receive***.
- **Localidade** trata da frequência dos acessos locais em oposição aos feitos em *hosts* remotos.
- Assim como concorrência e escalabilidade, **localidade** é um requisito fundamental em programação paralela.
- Importância da localidade é proporcional à relação do custo do acesso remoto sobre os acessos locais.
- Desempenho do computador, da rede utilizada e dos mecanismos de acesso para envio e recebimento de dados pela rede influenciam nessa relação.

Interconexão Física

- Redes de interconexão rápidas são decisivas em sistemas paralelos e *clusters*.
- Escalabilidade de aplicações em *clusters* era limitada pela alta latência das redes *Ethernet*, predominante em muitos cenários.
- Embora (10) *Gigabit Ethernet* seja amplamente utilizado atualmente, com largura de banda teórica de(10x) 125 MB/s, sua aplicação restringe-se aos casos em que a latência dos acessos não é fundamental.
- Outras soluções especializadas e com alto desempenho para interconexão de nós de processamento:
  - Infiniband
  - Myrinet
  - QsNet
  - SCI (Scalable Coherent Interface)


Architecture Share Over Time
1993-2011
Systems
500
400
300
200
100
0
06/1993
06/1994
06/1995
06/1996
06/1997
06/1998
06/1999
06/2000
06/2001
06/2002
06/2003
06/2004
06/2005
06/2006
06/2007
06/2008
06/2009
06/2010
06/2011
TOP500 Releases
MPP
Cluster
SMP
Constellations
Single Processor
Others


Interconnect Share Over Time
1993-2011
Systems
500
400
300
200
100
0
06/1993
06/1994
06/1995
06/1996
06/1997
06/1998
06/1999
06/2000
06/2001
06/2002
06/2003
06/2004
06/2005
06/2006
06/2007
06/2008
06/2009
06/2010
06/2011
TOP500 Releases
N/A
Gigabit Ethernet
SP Switch
HyperPlex
Infiniband
Hypercube
3-D Torus
Myrinet
Multi-stage crossbar
Full distributed crossbar
Proprietary
Infiniband DDR
Hypercube, tree
2-D mesh (torus)
Quadrics