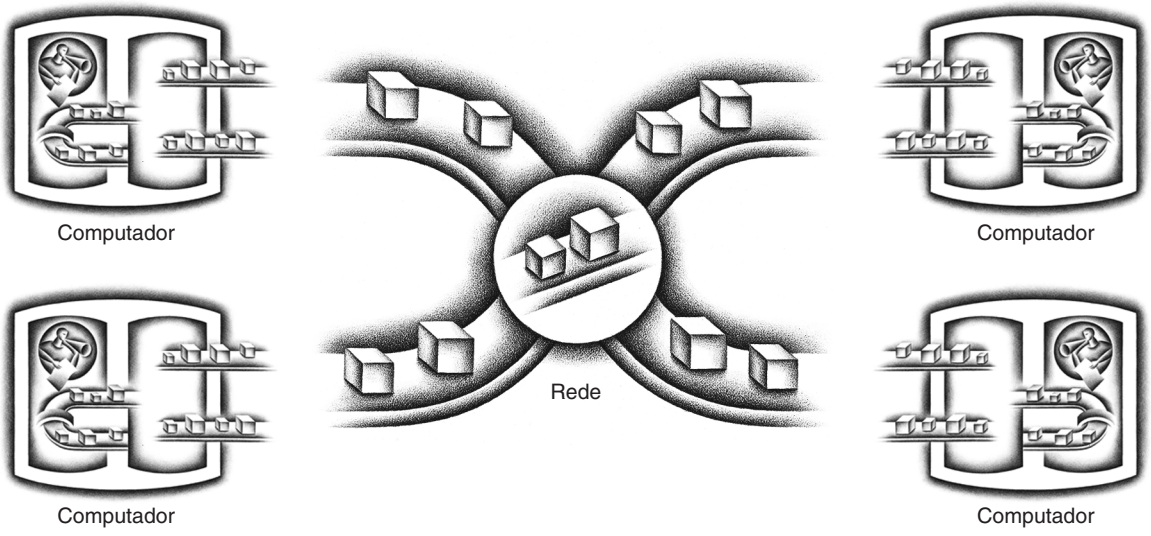
9

Multiprocessadores e Clusters

Existem, no mar, peixes mais finos
do que os que já foram pescados.
Provérbio irlandês

Os cinco componentes clássicos de um computador



"Sobre as montanhas da lua, pelo vale das sombras, cavalgue, cavalgue corajosamente." Respondeu a sombra: "Se você procurar o Eldorado!"

Edgar Allan Poe, "Eldorado", stanza 4, 1849

9.1 Introdução

Há muito tempo, os arquitetos de computadores têm buscado o El Dorado do projeto de computadores: criar computadores poderosos simplesmente conectando muitos computadores menores existentes. Essa visão dourada é a origem dos **multiprocessadores**. O cliente pede tantos processadores quantos seu orçamento permitir e recebe uma quantidade correspondente de desempenho. Portanto, os multiprocessadores podem ser escaláveis: o hardware e o software são projetados para serem vendidos com um número variável de processadores, com algumas máquinas variando por um fator de mais de 50. Como o software é escalável, alguns multiprocessadores podem suportar operar mesmo com a ocorrência de quebras no hardware; ou seja, se um único processador falhar em um multiprocessador com n processadores, o sistema fornece serviço continuado com $n - 1$ processadores. Finalmente, os multiprocessadores possuem o desempenho absoluto mais alto – mais rápido do que o uniprocessador mais rápido que existe.

multiprocessador Processadores em paralelo com um único endereço compartilhado

cluster Um conjunto de computadores conectados por uma rede local (LAN) que funciona como um único e grande multiprocessador.

A boa notícia é que o multiprocessador estabeleceu uma cabeça de ponte. Tendo em mente que o microprocessador é hoje o processador mais econômico, é consenso que se você não pode tratar um workload em um microprocessador, então, um multiprocessador ou um **cluster** composto de muitos microprocessadores é mais econômico do que construir um processador de alto desempenho por meio de uma tecnologia exótica. Há muitas aplicações científicas em que é extremamente difícil fazer progresso com um único microprocessador: previsão de tempo, cruzamento de proteínas e mesmo pesquisa de inteligência extraterrestre. Portanto, a Figura 9.1.1 mostra que a indústria da computação de alto desempenho depende dos multiprocessadores e dos clusters.

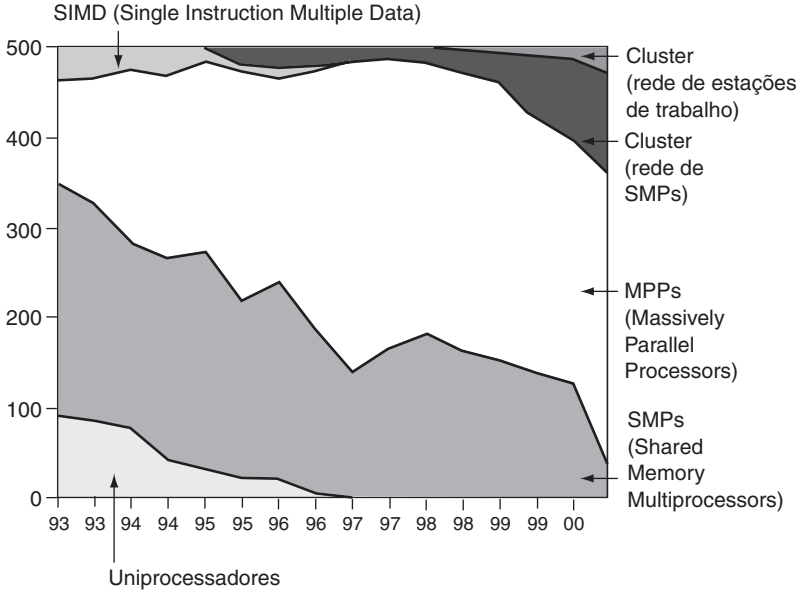


FIGURA 9.1.1 Representação dos primeiros 500 supercomputadores em uma década. Os números para 1993/1998/2003 são 93/0/0 para uniprocessadores, 251/175/0 para SMPs, 121/310/165 para MPPs, 35/0/0 para SIMDs, 0/14/127 para clusters de SMPs e 0/1/208 para clusters de estações de trabalho. Observe que nos últimos cinco anos, os uniprocessadores, os SMPs e os SIMDs desapareceram, enquanto os clusters de vários tipos cresceram de 3% para 67%. Além disso, a maioria dos MPPs na lista parece semelhante aos clusters. O desempenho é medido como a velocidade de execução do Linpack, que resolve um denso sistema de equações lineares. Essa lista em *www.top500.org* é atualizada duas vezes por ano. Esse site usa o termo *constelação* para se referir a uma rede de servidores SMP e o termo *cluster* para indicar um cluster de PCs ou estações de trabalho, que podem ser uniprocessadores ou pequenos SMPs. Essa vaga distinção não é usada neste texto; neste livro, um cluster é uma coleção de computadores conectados por uma rede local padrão usada para uma tarefa comum.

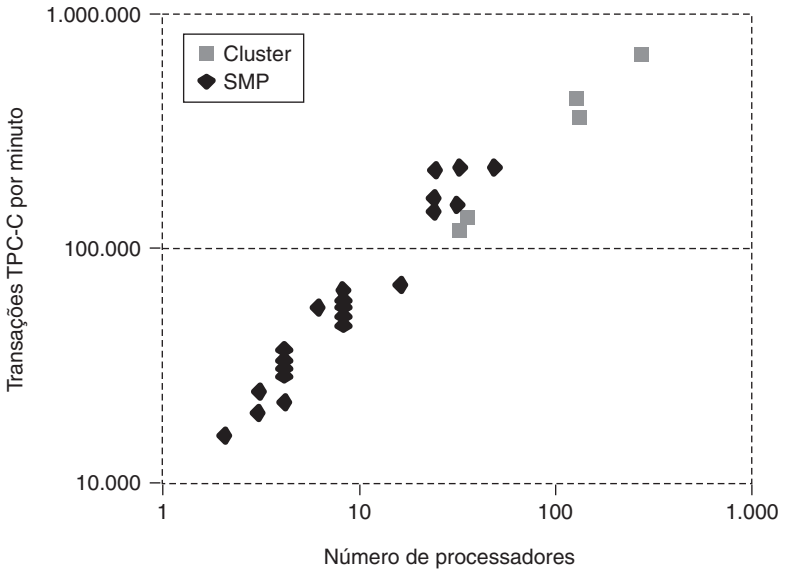


FIGURA 9.1.2 Desempenho *versus* número de processadores para TPC-C em uma escala log-log. Essas representações são de computadores executando a versão 5 do benchmark TPC-C. Repare que mesmo o menor computador possui 2 processadores. Os clusters obtêm alto desempenho escalando-os. Eles podem sustentar 2.500 a 3.800 transações por minuto por processador de 32 a 280 processadores. Os clusters não só possuem a maior taxa tpmC, mas eles apresentam uma relação custo-desempenho ($/tpmC) melhor do que qualquer SMP com um custo total de mais de US$1 milhão.

Também existem aplicações fora das ciências que são complexas: mecanismos de busca, servidores Web e bancos de dados. Por exemplo, a Figura 9.1.2 ilustra que a indústria de bancos de dados foi padronizada em multiprocessadores e clusters. Conseqüentemente, eles agora incorporam um mercado significativo.

Como os processadores operando em paralelo normalmente compartilham dados, eles também precisam coordenar quando estão operando em dados compartilhados; caso contrário, um processador poderia começar a processar dados antes que outro tenha acabado de processá-los. Essa coordenação é chamada **sincroni ação** . Quando o compartilhamento é suportado por um único espaço de endereçamento, precisa haver um mecanismo separado para sincronização. Um método é usar um **loc** (bloqueio): apenas um processador de cada vez pode adquirir o lock, e outros processadores interessados em compartilhar dados precisam esperar até que o processador original desbloqueie a variável. O lock é descrito na Seção 9.3.

Os multiprocessadores e clusters comerciais normalmente definem alto desempenho como alta vazão para tarefas independentes. Essa definição está em contraste com executar uma única tarefa em múltiplos processadores simultaneamente.

Eis algumas perguntas que norteiam os projetos de multiprocessadores e clusters:

- Como os processadores paralelos compartilham dados?
- Como os processadores paralelos se coordenam?
- Quantos processadores?

As respostas à primeira pergunta fazem parte de dois campos principais. Os processadores com um *único espaço de endereçamento*, algumas vezes chamados de processadores de **memória compartilhada**, oferecem ao programador um único espaço de endereçamento de memória que todos os processadores compartilham. Os processadores se comunicam por meio de variáveis compartilhadas na memória, com todos os processadores sendo capazes de acessar qualquer local da memória por meio de loads e stores.

sincronização O processo de coordenar o comportamento de dois ou mais processos, que podem estar sendo executados em diferentes processadores.

lock Um dispositivo de sincronização que permite acesso a dados apenas a um processador de cada vez.

memória compartilhada Uma memória para um processador paralelo com um único espaço de endereçamento, envolvendo comunicação implícita com loads e stores.

SMP (Symmetric Multiprocessors) ou **UMA (Uniform Memory Access)** Um multiprocessador em que os acessos à memória principal levam o mesmo período, independente do processador que requisita o acesso e de qual word é requisitada.

NUMA (Nonuniform Memory Access) Um tipo de multiprocessador de espaço de endereçamento único em que alguns acessos à memória são mais rápidos que outros, dependendo de que processador requisita que word.

troca de mensagens Comunicação entre vários processadores explicitamente enviando e recebendo informações.

rotina de envio de mensagens Uma rotina usada por um processador em máquinas com memórias privadas, com o objetivo de passar dados para outro processador.

rotina de recepção de mensagens Uma rotina usada por um processador em máquinas com memórias privadas, com o objetivo de aceitar uma mensagem de outro processador.

Os multiprocessadores de espaço de endereçamento único podem ser de dois estilos. O primeiro leva o mesmo tempo para acessar a memória principal independente de qual processador o requisita e de qual word é requisitada. Essas máquinas são chamadas **UMA (Uniform Memory Access)** ou **SMP (Symmetric Multiprocessors)**. No segundo estilo, alguns acessos à memória são mais rápidos do que outros, dependendo de que processador pede que word. Essas máquinas são chamadas **NUMA (Nonuniform Memory Access)**. Como você poderia esperar, as dificuldades de programação em um multiprocessador NUMA são diferentes das dificuldades em um multiprocessador UMA, mas as máquinas NUMA podem escalar para tamanhos maiores e, portanto, potencialmente possuem desempenho mais alto. A Figura 9.1.3 mostra o número de processadores e tempos de acesso à memória não local para SMPs e NUMAs comerciais.

O modelo alternativo para comunicação usa a **troca de mensagens** para comunicação entre processadores. A troca de mensagens é necessária para máquinas com *memórias privadas*, em contraste com a memória compartilhada. Um exemplo é um cluster, que processadores em diferentes computadores desktop se comunicam trocando mensagens por meio de uma rede local. Desde que o sistema tenha **rotinas de envio e recepção de mensagens**, a coordenação está embutida na troca de mensagens, já que um processador sabe quando uma mensagem é enviada e o processador receptor sabe quando uma mensagem chega. O processador receptor, então, pode enviar uma mensagem de volta para o emissor dizendo que a mensagem chegou se o emissor precisar dessa confirmação.

Além desses dois estilos de comunicação principais, os multiprocessadores são construídos em duas organizações básicas: os processadores conectados por um único barramento e os processadores conectados por uma rede. O número de processadores no multiprocessador tem muito a ver com essa escolha. Examinaremos esses dois estilos em detalhes nas Seções 9.3 e 9.4.

Começaremos analisando os aspectos gerais da programação em multiprocessadores.

Colocando em perspectiva

A Figura 9.1.4 mostra a relação entre o número de processadores em um multiprocessador e a escolha da comunicação por endereçamento compartilhado ou da comunicação por troca de mensagens e a escolha da conexão de barramento ou da conexão física de rede. A comunicação de endereçamento compartilhado é subdividida em acesso à memória uniforme e não uniforme. Embora haja muitas escolhas para alguns números de processadores, para outras regiões existe um amplo consenso.

Multiprocessador	Ano de lançamento	SMP ou NUMA	Máximo de processadores	Rede de interconexão	Tempo típico de acesso à memória remota (ns)
Servidores Sun Starfire	1996	SMP	64	Barramentos de endereço múltiplos, switch de dados	500
SGI Origin 3000	1999	NUMA	512	Hipercubo largo	500
Cray T3E	1996	NUMA	2048	Toro 3D de duas vias	300
Série HP V	1998	SMP	32	Crossbar 8 × 8	1000
Compaq AlphaServer GS	1999	SMP	32	Barramentos comutados	400
Sun V880	2002	SMP	8	Barramentos comutados	240
HP Superdome 9000	2003	SMP	64	Barramentos comutados	275

Figura 9.1.3 Tempos de acesso remoto típicos para recuperar uma word de uma memória remota em multiprocessadores de memória compartilhada.

Categoria	Escolha		Número de processadores
Modelo de comunicação	Troca de mensagens		8-2048
	Endereçamento compartilhado	NUMA	8-256
		UMA	2-64
Conexão física	Rede		8-256
	Barramento		2-36

FIGURA 9.1.4 Opções no estilo de comunicação e na conexão física para multiprocessadores conforme varia o número de processadores. Observe que o espaço de endereçamento compartilhado é dividido em máquinas de acesso à memória uniforme (UMA) e acesso à memória não uniforme (NUMA).

9.2 Programando multiprocessadores

Uma grande preocupação freqüentemente expressa em conexão com máquinas de computação bastante rápidas... é que elas... ficarão sem trabalho. ...Precisa ser considerado que... o tamanho do problema era determinado pela velocidade das máquinas de computação então disponíveis. ...Para máquinas mais rápidas, o mesmo mecanismo automático exercerá pressão contra problemas de maior tamanho.

John von Neumann, palestra apresentada no seminário da IBM sobre computação científica, em novembro de 1949.

A má notícia é que ainda não se sabe quantas aplicações importantes serão executadas mais rapidamente em multiprocessadores por meio de processamento paralelo. O obstáculo não é o preço do uniprocessador usado para compor multiprocessadores, as falhas nas topologias das redes de interconexão ou a indisponibilidade de linguagens de programação apropriadas; a dificuldade tem sido que muito poucos programas de aplicação têm sido reescritos para completar tarefas mais rapidamente em multiprocessadores. Como é mais difícil ainda encontrar aplicações que possam tirar proveito de muitos processadores, o problema é maior para multiprocessadores de grande escala.

Haja vista a dificuldade de programação, a maioria das histórias de sucesso no processamento paralelo é um resultado de magos de software desenvolvendo um subsistema paralelo que apresenta uma interface seqüencial. Exemplos incluem bancos de dados, servidores de arquivos, pacotes de projeto auxiliados por computador e sistemas operacionais de multiprocessamento.

Mas por que isso acontece? Por que os programas de processamento paralelo devem ser tão mais difíceis de desenvolver do que os programas seqüenciais?

A primeira razão é que você *precisa* obter um bom desempenho e eficiência do programa paralelo em um multiprocessador; caso contrário, você usaria um uniprocessador, já que a programação é mais fácil. Na verdade, as técnicas de projeto de uniprocessadores, como execução superescalar e fora de ordem, tiram vantagem do paralelismo em nível de instrução, normalmente sem envolvimento do programador. Tal inovação reduz a necessidade de reescrever programas para multiprocessadores.

Por que é difícil escrever programas de multiprocessador que sejam rápidos, especialmente quando o número de processadores aumenta? Como uma analogia, pense no overhead de comunicação para uma tarefa realizada por uma pessoa comparado com o overhead para uma tarefa realizada por um comitê, especialmente enquanto o tamanho do comitê aumenta. Embora n pessoas possam ter o potencial para concluir qualquer tarefa n vezes mais rápido, o overhead de comunicação para o grupo pode impedir isso; a multiplicação da velocidade por n se torna especialmente improvável conforme n aumenta. (Imagine a mudança no overhead de comunicação se um comitê crescer de 10 para 1.000 pessoas e, depois, para 1.000.000.)

Outra razão por que é difícil escrever programas paralelos é que o programador precisa saber muito sobre o hardware. Em um uniprocessador, o programador de linguagem de alto nível escreve o programa ignorando grande parte da organização básica da máquina – esse é o trabalho do compilador. Infelizmente, isso não é tão simples para multiprocessadores.

Embora esse segundo obstáculo esteja começando a diminuir, nossa discussão no Capítulo 4 revela um terceiro obstáculo: a Lei de Amdahl. Ela nos lembra que mesmo as pequenas partes de um programa precisam estar em paralelo para alcançar seu completo potencial; assim, aproximar-se do speedup linear envolve descobrir novos algoritmos que sejam inerentemente paralelos.

PROBLEMA DO SPEEDUP

EXEMPLO Suponha que você queira alcançar speedup linear com 100 processadores. Que fração da computação original pode ser seqüencial?

RESPOSTA A Lei de Amdahl (página 202) diz que:

Tempo de execução após melhoria =

$$\frac{\text{Tempo de execução afetado pela melhoria}}{\text{Quantidade de melhoria}} + \text{Tempo de execução não afetado}$$

Substituir pela meta de speedup linear com 100 processadores significa que o tempo de execução é reduzido por 100:

$$\frac{\text{Tempo de execução antes da melhoria}}{100} =$$

$$\frac{\text{Tempo de execução afetado pela melhoria}}{100} + \text{Tempo de execução não afetado}$$

Já que

Tempo de execução antes da melhoria =

Tempo de execução afetado pela melhoria + Tempo de execução não afetado

Se substituirmos isso na equação acima, teremos:

$$\frac{\text{Tempo de execução afetado pela melhoria} + \text{Tempo de execução não afetado}}{100} = \frac{\text{Tempo de execução afetado pela melhoria}}{100} + \text{Tempo de execução não afetado}$$

Simplificando, temos:

$$\frac{\text{Tempo de execução não afetado pela melhoria}}{100} = \text{Tempo de execução não afetado}$$

Isso só pode ser verdadeiro se o Tempo de execução não afetado for 0.

Portanto, para obter speedup linear com 100 processadores, *nenhuma* computação original pode ser seqüencial. Dito de outra forma, obter uma speedup de 99 dos 100 processadores significa que a porcentagem do programa original que era seqüencial precisaria ser de 0,01% ou menos.

Entretanto, existem aplicações com um substancial paralelismo.

PROBLEMA DO SPEEDUP, AINDA MAIOR

EXEMPLO

Suponha que você queira realizar duas somas: uma é a soma de duas variáveis escalares e outra é uma soma matricial de um par de arrays bidimensionais de tamanho 1.000 por 1.000. Que speedup você obtém com 1.000 processadores?

RESPOSTA

Se considerarmos que o desempenho é uma função do tempo para uma adição, t, então há 1 adição que não se beneficia dos processadores paralelos e 1.000.000 adições que se beneficiam. Se o tempo antes é de 1.000.001t,

$$\text{Tempo de execução após melhoria} = \frac{\text{Tempo de execução afetado pela melhoria}}{\text{Quantidade de melhoria}} + \text{Tempo de execução não afetado}$$

$$\text{Tempo de execução após melhoria} = \frac{1.000.000t}{1.000} + 1t = 1.001$$

Então, o speedup é:

$$\text{Speedup} = \frac{1.000.001}{1.001} = 999$$

Mesmo se a parte seqüencial expandisse para 100 somas de variáveis escalares em relação a uma soma de um par de arrays de 1.000 por 1.000, o speedup ainda seria 909.

9.3 Multiprocessadores conectados por um único barramento

O alto desempenho e o baixo custo do microprocessador inspiraram um novo interesse nos multiprocessadores na década de 1980. Vários microprocessadores podem ser convenientemente colocados em um barramento comum por várias razões:

- Como cada microprocessador é muito menor do que um processador multichip, mais processadores podem ser colocados em um barramento.
- As caches podem reduzir o tráfego de barramento.
- Foram inventados mecanismos para manter as caches e a memória consistentes para multiprocessadores, assim como as caches e a memória são mantidas consistentes para a E/S, simplificando, assim, a programação.

A Figura 9.3.1 é um esboço de um multiprocessador de barramento único genérico.

O tráfego por processador e a largura de banda de barramento determinam o número útil de processadores nesse multiprocessador. As caches duplicam os dados em suas memórias mais rápidas para reduzir a latência para os dados *e* para reduzir o tráfego de memória no barramento.

PROGRAMA PARALELO (BARRAMENTO ÚNICO)

EXEMPLO

Suponha que queremos somar 100.000 números em um computador com multiprocessador de barramento único. Vamos considerar que temos 100 processadores.

RESPOSTA

A primeira etapa novamente seria dividir o conjunto de números em subconjuntos do mesmo tamanho. Não alocamos os subconjuntos em memórias diferentes, já que existe uma única memória para essa máquina; apenas atribuímos endereços iniciais diferentes a cada processador. `Pn` é o número do processador, entre 0 e 99. Todos os processadores começam o programa executando um loop que soma seu subconjunto de números:

```
sum[Pn] = 0;
for (i = 1000*Pn; i < 1000*(Pn+1); i = i + 1)
   sum[Pn] = sum[Pn] + A[i]; /* soma as áreas atribuídas */
```

A próxima etapa é somar essas muitas somas parciais; portanto, dividimos para conquistar. A metade dos processadores soma pares de somas parciais, depois, um quarto soma pares das novas somas parciais e assim por diante, até que tenhamos uma única soma final. Desejamos que cada processador tenha sua própria versão da variável contadora do loop `i`, então temos de indicar que ela é uma variável "privada".

Neste exemplo, os dois processadores precisam ser sincronizados antes que o processador "consumidor" tente ler o resultado do local da memória escrito pelo processador "produtor"; caso contrário, o consumidor pode ler o valor antigo dos dados. Aqui está o código (`half` também é privada):

```
half = 100; /* 100 processadores no multiprocessador */
do
{
         synch( ); /* espera a conclusão da soma parcial */
         if (half%2 != 0 && Pn == 0)
               sum[0] = sum[0] + sum[half-1];
               /* soma condicional necessária quando half é
               ímpar; Processor0 obtém elemento ausente */
         half = half/2; /* linha divisora sobre quem soma */
         if (Pn < half) sum[Pn] = sum[Pn] + sum[Pn+half];
}
while (half == 1); /* sai com soma final em Sum[0] */
```

coerência de cache Consistência no valor dos dados entre as versões nas caches de vários processadores.

Lembre-se do Capítulo 8 de que E/S pode experimentar inconsistências no valor dos dados entre a versão na memória e a versão na cache. Esse problema de **coerência de cache** se aplica tanto a multiprocessadores quanto a E/S. Diferente da E/S, que raramente usa múltiplas cópias de dados (uma situação a ser evitada sempre que possível), como a segunda metade do exemplo sugere, múltiplos processadores normalmente requerem cópias dos mesmos dados em múltiplas caches. Como alternativa, os acessos a dados compartilhados podem ser forçados a sempre passar da cache para a memória, mas isso seria muito lento e exigiria muita largura de banda do barramento; o desempenho de um programa para multiprocessador depende do desempenho do sistema quando está compartilhando dados.

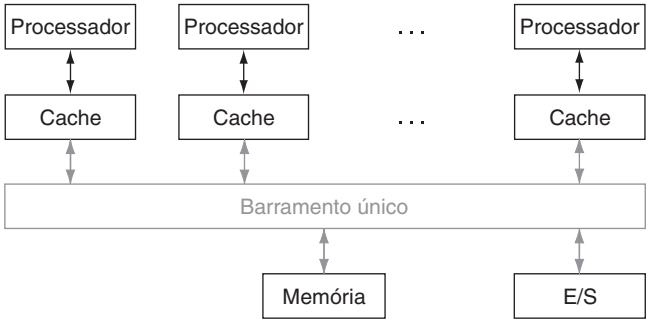


FIGURA 9.3.1 Um multiprocessador de barramento único. O tamanho típico é entre 2 e 32 processadores.

Os protocolos para manter coerência em múltiplos processadores são chamados *protocolos de coerência de cache*. As próximas subseções explicam os protocolos de coerência de cache e os métodos de sincronização de processadores usando coerência de cache.

Coerência de cache em multiprocessadores

coerência de cache com snooping Um método para manter coerência de cache em que todos os controladores de cache monitoram o barramento para determinar se possuem ou não uma cópia do bloco desejado.

O protocolo mais comum para manter a coerência de cache é chamado **snooping**. A Figura 9.3.2 mostra como as caches acessam a memória por meio de um barramento comum. Todos os controladores de cache monitoram o barramento para determinar se possuem uma cópia do bloco compartilhado.

O snooping se tornou popular com muitas máquinas da década de 1980, que usavam barramentos únicos para suas memórias principais. Esses uniprocessadores foram estendidos acrescentando processadores múltiplos nesse barramento para fornecer acesso fácil à memória compartilhada. As caches foram, então, acrescentadas para melhorar o desempenho de cada processador, levando a esquemas para manter as caches atualizadas pelo monitoramento (snooping) das informações nesse barramento compartilhado.

A manutenção da coerência possui dois componentes: leituras e escritas. Múltiplas cópias não são um problema durante a leitura, mas um processador precisa ter acesso exclusivo para escrever uma word. Os processadores também precisam ter a cópia mais recente quando estão lendo um objeto e, então, todos os processadores precisam obter valores novos após uma escrita. Portanto, os protocolos de snooping precisam localizar todas as caches que compartilham um objeto a ser escrito. A conseqüência de uma escrita em dados compartilhados é invalidar todas as outras cópias ou atualizar as cópias compartilhadas com o valor sendo escrito.

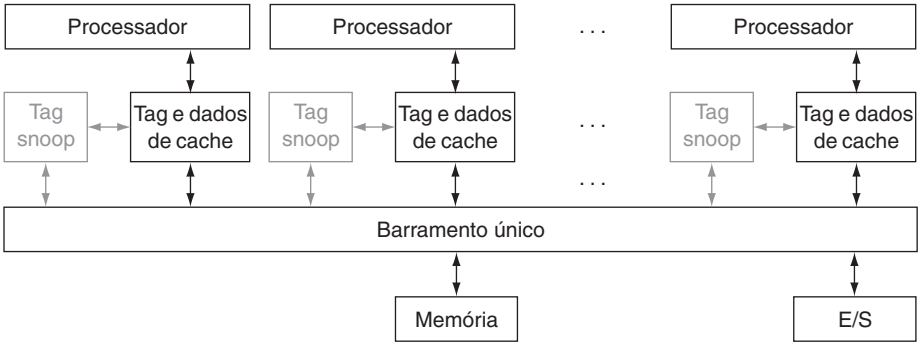


FIGURA 9.3.2 Um multiprocessador de barramento único usando coerência de cache com snooping. O conjunto de tags extra, mostrado em destaque, é usado para manipular requisições de snoop. As tags são duplicadas para reduzir as demandas de snooping nas caches.

Os bits de status já existentes em um bloco de cache são expandidos para protocolos de snooping, e essas informações são usadas no monitoramento de atividades de barramento. Em uma falha de leitura, todas as caches verificam se elas possuem uma cópia do bloco requisitado e, depois, tomam a ação apropriada, como fornecer os dados para a cache que falhou. Da mesma forma, em uma escrita, todas as caches verificam se possuem uma cópia e, então, agem, invalidando ou atualizando sua cópia com o novo valor.

Como cada transação de barramento verifica as tags de endereço de cache, você precisa considerar que ela interfere no processador. Ela interferiria, se não para duplicar a parte da tag de endereço da cache – não a cache inteira, para ter uma porta de leitura extra para snooping (veja a Figura 9.3.2). Desse modo, o snooping raramente interfere com o acesso do processador à cache. Quando há interferência, o processador provavelmente será suspenso porque a cache está indisponível.

write-invalidate Um tipo de protocolo de snooping em que o processador de escrita faz com que todas as cópias em outras caches sejam invalidadas antes de mudar sua cópia local, o que permite atualizar os dados locais até que outro processador os solicite.

Os multiprocessadores comerciais baseados em cache usam caches write-back porque ele reduz o tráfego de barramento e, portanto, permite mais processadores em um único barramento. Para preservar essa preciosa largura de banda de comunicações, todas as máquinas usam **write-invalidate** como o protocolo de coerência padrão: o processador de escrita faz com que todas as cópias em outras caches sejam invalidadas antes de mudar sua cópia local; depois, eles estão liberados para atualizarem os dados *locais* até que outro processador os solicite. O processador de escrita ativa um sinal de invalidação no barramento e todas as caches verificam se elas possuem uma cópia; se possuírem, elas precisam invalidar o bloco contendo a word. Portanto, esse esquema permite diversos leitores mas apenas um escritor.

As medições até agora indicam que os dados compartilhados possuem menos localidade espacial e temporal do que outros tipos de dados. Assim, as falhas de dados compartilhados freqüentemente dominam o comportamento da cache, ainda que possam representar apenas de 10% a 40% dos acessos a dados. A Figura 9.3.3 mostra a fração das falhas devido à coerência enquanto o número de processadores varia em um SMP.

Interface hardware/software

falso compartilhamento Uma situação de compartilhamento em que duas variáveis compartilhadas não relacionadas estão localizadas no mesmo bloco de cache e o bloco inteiro é trocado entre processadores ainda que os processadores estejam acessando variáveis diferentes.

Um princípio é o de que o tamanho do bloco desempenha um importante papel na coerência de cache. Por exemplo, considere o caso de snooping em uma cache com um tamanho de bloco de oito words, com uma única word escrita e lida alternadamente por dois processadores. Um protocolo que apenas difunde ou envia uma única word tem uma vantagem sobre um que transfere o bloco inteiro.

Blocos grandes também podem causar o que se chama **falso compartilhamento**: Quando duas variáveis compartilhadas não relacionadas estão localizadas no mesmo bloco de cache, o bloco inteiro é trocado entre processadores mesmo que os processadores estejam acessando diferentes variáveis (veja os Exercícios 9.5 e 9.6). As pesquisas em compiladores estão reduzindo o falso compartilhamento alocando dados altamente correlacionados no mesmo bloco de cache e, portanto, reduzindo razões de falhas de cache.

Detalhamento: em um multiprocessador usando coerência de cache em um barramento único, o que acontece se dois processadores tentarem escrever na mesma word de dados compartilhada no mesmo ciclo de clock? O árbitro de barramento decide que processador toma o barramento primeiro e esse processador invalidará ou atualizará a cópia do outro processador, dependendo do protocolo. O segundo processador, então, realiza sua escrita. A arbitração de barramento força o comportamento seqüencial das escritas no mesmo bloco por diferentes processadores, e isso explica como as escritas de diferentes processadores em diferentes words no mesmo bloco funcionarão corretamente.

A política de quando um processador vê uma escrita de outro processador é chamada de *modelo de consistência de memória*. O mais conservador é chamado de *consistência seqüencial*: o resultado de qualquer execução é o mesmo que se os acessos de cada processador fossem mantidos na ordem e os acessos entre diferentes processadores fossem intercalados. Algumas máquinas usam modelos mais liberais para atingir um desempenho de memória melhor.

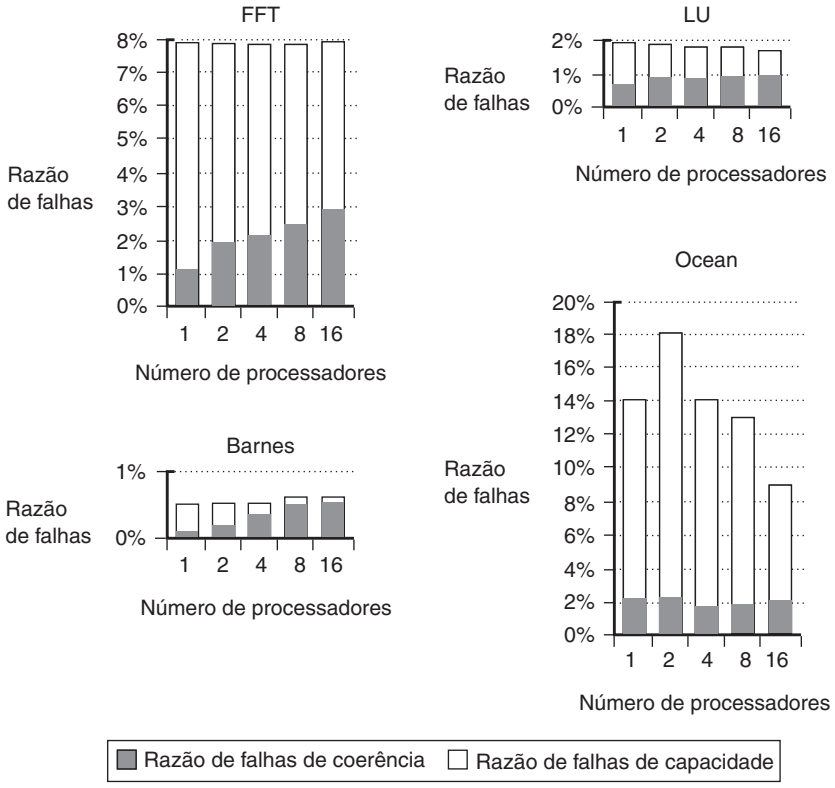


FIGURA 9.3.3 As razões de falhas de dados podem variar de maneiras não óbvias conforme o número de processadores é aumentado de 1 para 16. As razões de falhas incluem razões de falhas de coerência e de capacidade. Para todas essas execuções, o tamanho de cache é 64KB, associativo por conjunto de duas vias, com blocos de 32 bytes. Observe que a escala no eixo *y* para cada benchmark é diferente, de modo que o comportamento dos benchmarks individuais pode ser visto claramente. (Da Figura 6.23 em Hennessy e Patterson, *Arquitetura de computadores: Uma abordagem quantitativa*, terceira edição, 2003.)

Detalhamento: nosso exemplo usou uma primitiva de **sincronização por barreira**; os processadores esperam na barreira até que cada processador a tenha atingido. Então, eles continuam. A sincronização por barreira permite que todos os processadores sejam sincronizados rapidamente. Essa função pode ser implementada no software ou com a primitiva de sincronização de lock, descrita em breve.

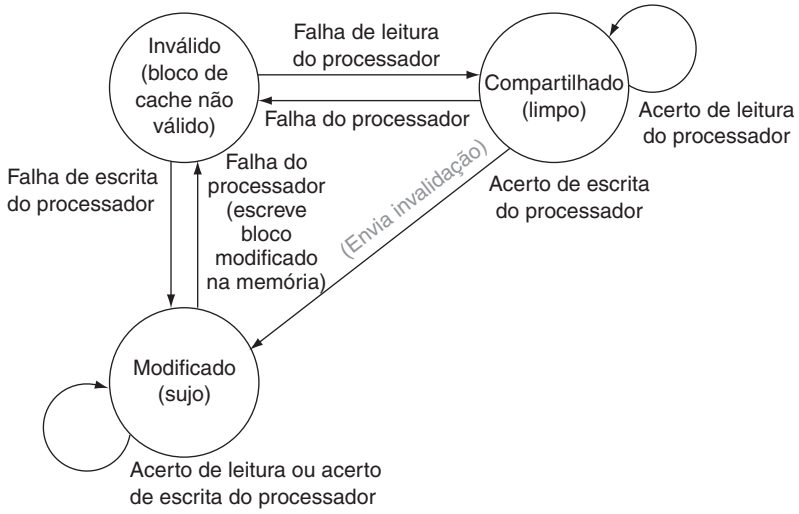
sincronização por barreira Um esquema de sincronização em que os processadores esperam na barreira e não continuam até que todos os processadores a tenham alcançado.

Um exemplo de protocolo de coerência de cache

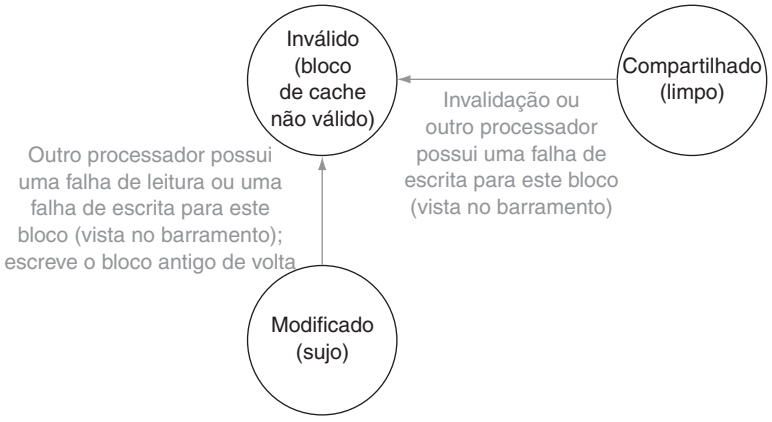
Para ilustrar as particularidades de um protocolo de coerência de cache, a Figura 9.3.4 mostra um diagrama de transição de estados finitos para um protocolo write-invalidate baseado em uma política write-back. Cada bloco de cache está em um de três estados:

1. *Compartilhado* (apenas leitura): esse bloco de cache é limpo (não escrito) e pode ser compartilhado.
2. *Modificado* (leitura/escrita): esse bloco de cache é sujo (escrito) e pode *não* ser compartilhado.
3. *Inválido*: esse bloco de cache não possui dados válidos.

Os três estados do protocolo são duplicados na figura para mostrar transições com base nas ações do processador em oposição às transições baseadas em operações de barramento. Essa duplicação é feita apenas para fins de ilustração; há realmente apenas uma máquina de estados finitos por bloco de cache, com estímulos vindo do processador anexado ou do barramento. Essa abstração também se



a. Transições de estado de cache usando sinais do processador



b. Transições de estado de cache usando sinais do barramento

FIGURA 9.3.4 Um protocolo write-invalidate de coerência de cache. A parte superior do diagrama mostra transições de estado baseadas em ações do processador associadas com essa cache; a parte inferior mostra transições baseadas em ações de outros processadores quando vistas como operações no barramento. Na verdade, há apenas uma máquina de estados em um bloco de cache, embora haja duas representadas aqui para esclarecer quando ocorre uma transição. As setas pretas e as ações especificadas em texto preto seriam encontradas em caches sem coerência; as setas e ações em destaque são incluídas para obter coerência de cache.

aplica a blocos de cache *não* presentes na cache; obviamente, essas máquinas de estados estão todas no estado inválido.

As transições no estado de um bloco de cache ocorrem em falhas de leitura, falhas de escrita ou acertos de escrita; os acertos de escrita não mudam o estado da cache. Vamos começar com uma falha de leitura, chamando o bloco a ser substituído de a *vítima*. Quando o processador tem uma falha de leitura, ele adquire o barramento e escreve a vítima se ela estiver no estado Modified (sujo). Todas as caches nos outros processadores monitoram a falha de leitura para ver se esse bloco está em sua cache. Se algum tiver uma cópia e ela estiver no estado Modified, então o bloco é escrito novamente e seu estado é alterado para o estado Invalid. (Alguns protocolos mudariam o estado para Shared.) A falha de leitura, então, é satisfeita lendo da memória, e o estado do bloco é definido em Shared. Note que o bloco é lido da memória se uma cópia estiver em uma cache ou não nesse protocolo.

As escritas são mais complexas. Vamos experimentar os acertos de escrita. Um acerto de escrita em um bloco Modified não causa nenhuma ação de protocolo. Um acerto de escrita em um bloco

Shared faz com que a cache adquira o barramento, envie um sinal de invalidação para bloquear quaisquer outras cópias, modifique a parte do bloco sendo escrita e mude o estado para Modified.

Por último estão as falhas de escrita. Uma falha de escrita em um bloco Shared em outra cache faz com que a cache adquira o barramento, envie um sinal de invalidação para bloquear todas as cópias, leia todo o bloco em que houve falha, modifique a parte do bloco sendo escrita e mude o estado para Modified.

Como você poderia imaginar, há muitas variações na coerência de cache muito mais complicadas do que esse modelo simples. A encontrada no Pentium 4 e em muitos outros microprocessadores é chamada **MESI**, um protocolo de invalidação de escrita cujo nome é um acrônimo para os quatro estados do protocolo: Modified (Modificado), Exclusive (Exclusivo), Shared (Compartilhado), Invalid (Inválido). Modified e Invalid são iguais aos Modificado e Inválido, descritos anteriormente. O estado Shared é o estado Compartilhado da Figura 9.3.4 dividido, dependendo de se existem múltiplas cópias (estado Shared) ou existe apenas uma (estado Exclusive). Em qualquer caso, a memória possui uma versão atualizada dos dados. Esse estado Exclusive extra significa que existe apenas uma cópia do bloco e, portanto, um acerto de escrita não precisa invalidar. Um acerto de escrita em um bloco Compartilhado na Figura 9.3.4 requer uma invalidação, já que pode haver diversas cópias.

protocolo de coerência de cache MESI Um protocolo write-invalidate cujo nome é um acrônimo para os quatro estados do protocolo: Modified, Exclusive, Shared, Invalid.

Outras variações nos protocolos de coerência incluem se as outras caches tentam fornecer o bloco se tiverem uma cópia, e se o bloco precisa ser invalidado em uma falha de leitura.

Sincronização usando coerência

Um dos principais requisitos de um multiprocessador de barramento único é ser capaz de coordenar processos que estejam trabalhando em uma tarefa comum. Em geral, um programador usará *variáveis de lock* (também conhecidas como *semáforos*) para coordenar ou sincronizar os processos. O desafio para o arquiteto de um multiprocessador é fornecer um mecanismo para decidir qual processador obtém o lock e fornecer a operação que bloqueia a variável. A arbitração é fácil para multiprocessadores de barramento único, já que o barramento é o único caminho até a memória: o processador que obtém o barramento bloqueia todos os outros processadores de ir à memória. Se o processador e o barramento fornecerem uma **operação de troca atômica**, os programadores poderão criar locks com a semântica correta. Aqui, o adjetivo *atômico* significa indivisível; portanto, uma troca atômica significa que o processador pode ler um local *e* atribuir a ele o valor bloqueado na mesma operação de barramento, evitando que qualquer outro processador ou dispositivo de E/S leia ou escreva na memória até que a troca seja concluída.

operação de troca atômica Uma operação em que o processador pode ler um local *e* atribuir a ele o valor bloqueado na mesma operação de barramento, evitando que qualquer outro processador ou dispositivo de E/S leia ou escreva na memória até que a troca seja concluída.

A Figura 9.3.5 mostra um procedimento típico para realizar um lock numa variável usando uma instrução de troca atômica. Considere que 0 significa unlocked (“siga”) e 1 significa locked (“pare”). Um processador primeiro lê a variável de lock para testar seu estado. O processador continua lendo e testando até que o valor indique unlocked. O processador, então, disputa com todos os outros processadores que estavam da mesma maneira (*spin waiting*) para ver quem pode realizar o lock na variável primeiro. Todos os processadores usam uma instrução de troca atômica que lê o valor antigo e armazena um 1 (“pare”) na variável de lock. O único vencedor verá o 0 (“siga”) e os perdedores verão um 1 que foi colocado lá pelo vencedor. (Os perdedores continuarão a escrever a variável com o valor locked de 1, mas isso não muda seu valor.) O processador vencedor, então, executa o código que atualiza os dados compartilhados. Quando o vencedor sai, ele armazena um 0 na variável de lock, iniciando, assim, a corrida novamente.

O MIPS não inclui uma instrução de troca atômica. Uma alternativa é ter um par de instruções no qual a segunda instrução retorna um valor do qual pode-se deduzir se o par de instruções foi executado *como se* as instruções fossem atômicas. O par de instruções é efetivamente atômico se parecer que todas as outras operações executadas por qualquer processador ocorrem antes ou após o par. Portanto, quando um par de instruções for efetivamente atômico, nenhum outro processador pode mudar o valor entre o par de instruções.

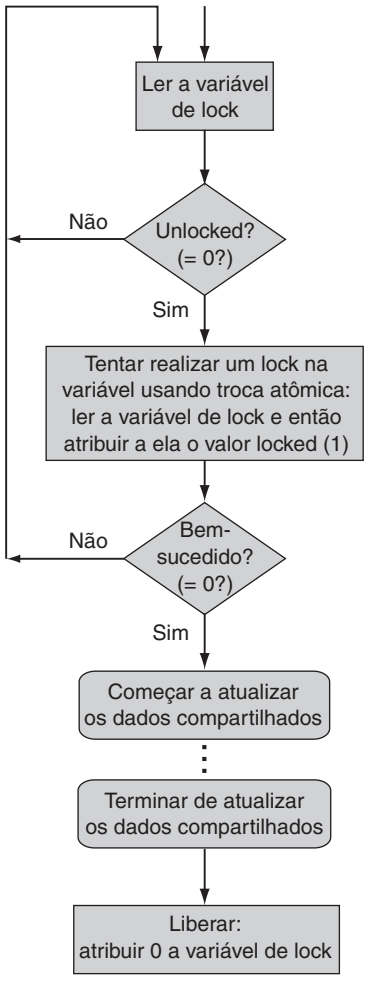


FIGURA 9.3.5 Etapas para adquirir um lock ou semáforo para sincronizar processos e, depois, para liberar o lock na saída da seção-chave do código.

O par MIPS de instruções inclui um load especial chamado *load linked* ou *load locked* (`ll`) e um store especial chamado *store conditional* (`sc`). Essas instruções são usadas em seqüência: se o conteúdo do local de memória especificado pelo load linked for alterado antes que ocorra o store conditional para o mesmo endereço, o store conditional falhará. Se o processador realizar uma troca de contexto entre as duas instruções, o store conditional também falhará. O store conditional é definido para retornar um valor indicando se o store foi bem-sucedido. Como o load linked retorna o valor inicial e o store conditional retorna 1 se for bem-sucedido e 0 no caso contrário, a seqüência a seguir implementa uma troca atômica no local de memória especificado pelo conteúdo de `$t1`:

```
try:  mov   $t3,$t4     # move o valor de troca
      ll    $t2,0($t1)  # load linked
      sc    $t3,0($t1)  # store conditional altera $t3
      beqz  $t3,try     # desvia se store conditonal falhar (=0)
      nop               # (delayed branch)
      mov   $t4,$t2     # coloca o valor do load em $t4
```

No final dessa seqüência, o conteúdo de `$t4` e o local de memória especificado por `$t1` foram atomicamente trocados. A qualquer momento em que um processador intervém e modifica o valor na memória entre as instruções `ll` e `sc`, o `sc` retorna 0 em `$t3`, fazendo a seqüência de código tentar novamente.

Vamos examinar como o esquema de spin lock da Figura 9.3.5 funciona com coerência de cache baseada em barramento. Uma vantagem desse algoritmo é que ele permite que os processadores utilizem spin wait em uma cópia local do lock em suas caches. Isso reduz a quantidade de tráfego de barramento; a Figura 9.3.6 mostra as operações de barramento e cache para múltiplos processadores tentando realizar lock numa variável. Uma vez que o processador com o lock armazena um 0 no lock, todas as outras caches vêem esse store e invalidam sua cópia da variável lock. Depois, elas tentam obter o novo valor 0 para o lock . Esse novo valor inicia a disputa para ver quem pode obter o lock primeiro. O vencedor obtém o barramento e armazena um 1 no lock; as outras caches substituem sua cópia da variável lock contendo 0 por um 1. Esse valor indica que a variável já está locked e, portanto, eles precisam retornar ao teste e ao spinning.

Etapa	Processador P0	Processador P1	Processador P2	Atividade do barramento	Memória
1	Obtém o lock	Spin, testando se lock = 0	Spin, testando se lock = 0	Nenhuma	
2	Coloca lock em 0; envia invalidação pelo barramento	Spin, testando se lock = 0	Spin, testando se lock = 0	write-invalidate da variável de lock enviado de P0	
3		Falha de cache	Falha de cache	Barramento serve a falha de cache de P2	
4		(Espera a falha de cache)	(Espera a falha de cache)	Resposta à falha de cache de P2	Atualiza a memória com bloco de P0
5		(Espera a falha de cache)	Testa lock = 0; êxito	Barramento serve a falha de cache de P1	
6		Testa lock = 0; êxito	Tenta trocar; precisa de permissão de escrita	Resposta à falha de cache de P1	Responde à falha de cache de P1; envia variável lock
7		Tenta trocar; precisa de permissão de escrita	Envia invalidação para obter permissão de escrita	Barramento serve a invalidação de P1	
8		Falha de cache	Troca; lê lock = 0 e coloca-o em 1	Barramento serve a falha de cache de P1	
9		Troca; lê lock = 1 e coloca-o em 1; volta para spin	Responde à falha de cache de P1, envia lock = 1	Resposta à falha de cache de P2	

Figura 9.3.6 Etapas da coerência de cache e tráfego de barramento para três processadores, P0, P1 e P2. Essa figura considera coerência write-invalidate. P0 começa com o lock (etapa 1). P0 sai e libera o lock (etapa 2). P1 e P2 disputam para ver quem lê o valor unlocked durante a troca (etapas 3 a 5). P2 vence e entra na *região crítica* (etapas 6 e 7), enquanto P1 realiza spins e waits (etapas 7 e 8). A "região crítica" é o nome para o código entre o lock e o unlock. Quando P2 sai da região crítica, ele coloca o lock em 0, o que invalidará a cópia na cache de P1, reiniciando o processo.

A dificuldade desse esquema aumenta para muitos processadores devido ao tráfego de comunicação gerado quando o lock é liberado.

9.4 Multiprocessadores conectados por uma rede

Os projetos de barramento único são atraentes mas limitados porque as três características desejáveis do barramento são incompatíveis: alta largura de banda, baixa latência e grande extensão. Também há um limite para a largura de banda de um único módulo de memória anexado a um barramento. Portanto, um barramento único impõe limitações práticas ao número de processadores que podem ser conectados a ele. Atualmente, o maior número de processadores conectados a um único barramento em um computador comercial é 36, e esse número parece estar diminuindo com o tempo.

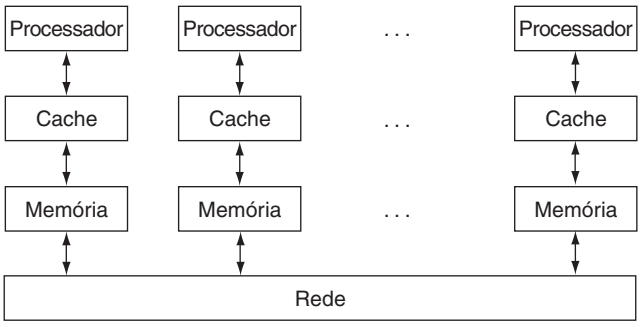


Figura 9.4.1 A organização de um multiprocessador conectado à rede. Observe que, diferente da Figura 9.3.1, a conexão do multiprocessador não está mais entre a memória e o processador.

Se o objetivo é conectar muitos mais processadores juntos, então, o projetista de computadores precisará usar mais do que um único barramento. A Figura 9.4.1 mostra como isso pode ser organizado. Observe que na Figura 9.3.1 o meio de conexão – o barramento – está entre os processadores e a memória, enquanto na Figura 9.4.1, a memória está anexada a cada processador e o meio de conexão – a rede – está entre esses nós combinados. Para sistemas de barramento único, o meio é utilizado em cada acesso à memória, enquanto no outro caso, ele é usado apenas para comunicação entre processadores.

memória compartilhada Uma memória para um processador paralelo com um único espaço de endereçamento, envolvendo comunicação implícita com loads e stores.

memória distribuída Memória física dividida em módulos, com alguns colocados próximos a cada processador em um multiprocessador.

CC-NUMA Um multiprocessador de acesso à memória não uniforme que mantém coerência para todas as caches.

Isso nos leva a um antigo debate sobre a organização da memória em processadores paralelos de grande escala. Infelizmente, o debate gira em torno de uma falsa dicotomia: **memória compartilhada** *versus* **memória distribuída**. A memória compartilhada realmente significa um único espaço de endereçamento, envolvendo comunicação implícita com loads e stores. O verdadeiro oposto de um endereçamento único são as *memórias privadas múltiplas*, envolvendo comunicação explícita com sends e receives.

As três categorias gerais de multiprocessadores são as seguintes:

- *UMA (Uniform Memory Access)*: O tempo de acesso à memória é o mesmo, independente de onde a word esteja localizada. Isso é normal com uma memória comum.
- *NUMA (Nonuniform Memory Access)*: O tempo de acesso à memória varia conforme seu endereço. Isso é normal com uma memória fisicamente distribuída.
- **CC-NUMA (Cache-coherent nonuniform memory access)**: Um NUMA com um mecanismo que garante a coerência de cache.

Em máquinas sem um espaço de endereçamento global único, a comunicação é explícita; o programador ou o compilador precisa enviar mensagens para transferir dados a outro nó e precisa receber mensagens para aceitar dados de outro nó.

PROGRAMA PARALELO (TROCA DE MENSAGENS)

EXEMPLO

Vamos experimentar nosso exemplo de soma novamente para um multiprocessador conectado à rede com 100 processadores usando múltiplas memórias privadas.

RESPOSTA

Como esse computador possui múltiplos espaços de endereçamento, o primeiro passo é distribuir os 100 subconjuntos para cada uma das memórias locais. O processador contendo os 100.000 números envia os subconjuntos para cada um dos 100 nós de memória de processador.

A próxima etapa é obter a soma de cada subconjunto. Essa etapa é simplesmente um loop que toda unidade de execução segue: ler uma word da memória local e adicioná-la a uma variável local:

```
sum = 0;
for (i = 0; i<1000; i = i + 1) /* loop em cada array */
 sum = sum + A1[i]; /* soma os arrays locais */
```

A última etapa é somar essas 100 somas parciais. A parte difícil é que cada soma parcial está localizada em uma unidade de execução diferente. Conseqüentemente, precisamos usar a rede de interconexão para enviar somas parciais para acumular a soma final. Em vez de enviar todas as somas parciais a um único processador, o que resultaria em acrescentar seqüencialmente as somas, mais uma vez dividimos para conquistar. Primeiro, metade das unidades de execução envia suas somas parciais para a outra metade das unidades de execução, onde duas somas parciais são somadas. Depois, um quarto das unidades de execução (metade da metade) envia essa nova soma parcial para o outro quarto das unidades de execução (a metade da metade restante) para a próxima rodada de somas. Essas divisões, envios e recepções continuam até haver uma única soma de todos os números. Seja `Pn` o número da unidade de execução, `send(x,y)` uma rotina que envia pela rede de interconexão para a unidade de execução número `x` o valor `y` e `receive( )` a função que recebe um valor da rede para essa unidade de execução:

```
limit = 100; half = 100; /* 100 processadores */
do
{
  half = (half+1)/2; /* linha divisória entre send e receive */
  if (Pn >= half && Pn < limit) send(Pn – half, sum);
  if (Pn < (limit/2)) sum = sum + receive( );
  limit = half; /* limite superior dos emissores */
}
while (half == 1); /* sai com a soma final */
```

Esse código divide todos os processadores em emissores ou receptores, e cada processador receptor recebe apenas uma mensagem, de modo que podemos presumir que um processador receptor estará suspenso até receber uma mensagem. Portanto, send e receive podem ser usados como primitivas para sincronização e para comunicação, já que os processadores estão cientes da transmissão dos dados.

Se houver um número ímpar de nós, o nó central não participa da emissão/recepção. O limite, então, é definido de modo que esse nó seja o nó mais alto na próxima iteração.

Endereçamento em processadores paralelos de grande escala

A maioria dos processadores comerciais de grande escala usa memória distribuída; de outro modo, é muito difícil ou muito caro construir uma máquina que possa ser escalada para muitos processadores com muitos módulos de memória.

Outro aspecto que envolve as máquinas de memória distribuída é a comunicação. Para o projetista de hardware, a solução mais simples é oferecer apenas send e receive em vez da comunicação implícita possível como parte de qualquer load ou store. Send e receive também possuem a vantagem de facilitar para o programador otimizar a comunicação: É mais simples unir computação com comunicação usando sends e receives explícitos do que com loads e stores implícitos.

Interface hardware/software

Acrescentar uma camada de software para fornecer um espaço de endereçamento único sobre sends e receives de modo que a comunicação seja possível como parte de qualquer load ou store é mais difícil, embora isso seja comparável ao sistema de memória virtual já encontrado na maioria dos processadores (veja o Capítulo 7). Na memória virtual, um uniprocessador usa tabelas de páginas para decidir se um endereço aponta para dados na memória local ou em um disco; esse sistema de tradução pode ser modificado para decidir se o endereço aponta para dados locais, para dados na memória de outro processador ou para o disco. Embora a *memória virtual compartilhada*, como é chamada, crie a ilusão de memória compartilhada – exatamente como a memória virtual cria a ilusão de uma memória muito grande –, já que ela chama o sistema operacional, o desempenho normalmente é tão lento que a comunicação de memória compartilhada precisa ser rara ou a maioria do tempo é gasta transferindo páginas.

Por outro lado, loads e stores normalmente têm um overhead de comunicação muito mais baixo do que sends e receives. Além disso, algumas aplicações terão referências a informações remotas acessadas apenas ocasional e imprevisivelmente, de modo que é muito mais eficiente usar um endereço para dados remotos quando *demandados* em vez de recuperá-los para o caso de *poderem* ser usados. Esse tipo de máquina possui **memória compartilhada distribuída (DSM)**.

memória compartilhada distribuída (DSM) Um esquema de memória que usa endereços para acessar dados remotos quando demandados em vez de recuperar os dados para o caso de precisarem ser usados.

As caches são importantes para o desempenho independente de como a comunicação seja realizada; assim, queremos permitir que os dados compartilhados apareçam na cache do processador que possui os dados bem como no processador que os requisitou. Portanto, o espaço de endereçamento global único em um multiprocessador conectado à rede reacende a questão da coerência de cache, já que há múltiplas cópias dos mesmos dados com o mesmo endereço em diferentes processadores. Claramente, os protocolos de snooping de barramento da Seção 9.3 não funcionarão aqui, pois não há um barramento único em que todas as referências de memória sejam broadcast. Como os projetistas do Cray T3E não tinham qualquer barramento para suportar coerência de cache, o T3E possui um único espaço de endereçamento mas sem coerência de cache.

diretório Um depósito de informações sobre o estado de cada bloco na memória principal, incluindo que caches possuem cópias do bloco, se ele é sujo etc. Usado para coerência de cache.

Uma alternativa com coerência de cache ao snooping de barramento são os **diretórios**. Nos protocolos baseados em diretórios, logicamente, um único diretório mantém o estado de cada bloco na memória principal. As informações no diretório podem incluir quais caches possuem cópias do bloco, se ele é sujo etc. Felizmente, as entradas de diretório podem ser distribuídas de modo que diferentes requisições possam ir para diferentes memórias, reduzindo, assim, a contenção e permitindo um projeto escalável. Os diretórios mantêm as características de que o status de compartilhamento de um bloco está sempre em um único local conhecido, tornando um processador paralelo de grande escala viável.

Os projetistas de caches com snooping e diretórios enfrentam problemas semelhantes; a única diferença é o mecanismo que detecta quando há uma escrita em dados compartilhados. Em vez de examinar o barramento para ver se existem requisições que exigem que a cache local seja atualizada ou invalidada, o controlador do diretório envia comandos explícitos a cada processador que tem uma cópia dos dados. Essas mensagens podem, então, ser enviadas pela rede.

Interface hardware/software

Observe que com um espaço de endereçamento único, os dados poderiam ser colocados arbitrariamente nas memórias de diferentes processadores. Isso tem duas conseqüências negativas para o desempenho. A primeira é que a penalidade de falha seria muito maior porque a requisição precisa atravessar a rede. A segunda é que a largura de banda de rede seria consumida movendo dados para os processadores corretos. Para programas que possuem baixas taxas de falhas, isso pode não ser signi-

ficante. Por outro lado, os programas com altas taxas de falhas terão desempenho muito menor quando os dados são aleatoriamente atribuídos.

Se o programador ou o compilador alocar dados para o processador que sejam improváveis de serem usados, então, essa armadilha de desempenho será removida. Diferente das organizações de memória privada, essa alocação precisa apenas ser boa, já que os dados em falha ainda podem ser buscados. Essa benevolência simplifica o problema da alocação.

Como o número de pinos por chip é limitado, nem todos os processadores podem ser conectados diretamente uns aos outros. Essa restrição inspirou todo um grupo de topologias a serem consideradas no projeto da rede. Na Seção 9.6, veremos as características de algumas das principais alternativas dos projetos de rede. Primeiro, vejamos outra maneira de conectar computadores em redes.

9.5 Clusters

Existem muitas aplicações de mainframe – como bancos de dados, servidores de arquivos, servidores Web, simulações e multiprogramação/processamento – apropriadas para serem executadas em máquinas conectadas mais livremente do que as máquinas NUMA com coerência de cache da seção anterior. Essas aplicações em geral precisam ser altamente disponíveis, exigindo algum tipo de tolerância a falhas e capacidade de reparação. Tais aplicações – além da semelhança dos nós de multiprocessador com os computadores desktop e o surgimento de redes locais de alta largura de banda baseadas em switches – são o motivo por que o processamento de grande escala usa *clusters* de computadores de uso geral facilmente encontrados no mercado.

Uma desvantagem dos clusters tem sido a de que o custo de administrar um cluster de n máquinas é aproximadamente igual ao custo de administrar n máquinas independentes, enquanto o custo de administrar um multiprocessador de espaço de endereçamento compartilhado com n processadores é aproximadamente igual ao de administrar uma única máquina.

Outro ponto negativo é que os clusters normalmente são conectados usando o barramento de E/S do computador, enquanto os multiprocessadores em geral são conectados no barramento de memória do computador. O barramento de memória possui largura de banda mais alta, permitindo que os multiprocessadores levem o link de rede a uma velocidade mais alta e tenham menos conflitos com o tráfego de E/S nas aplicações de E/S intensa.

Uma última desvantagem é a divisão da memória: um cluster de n máquinas possui n memórias independentes e n cópias do sistema operacional, mas um multiprocessador de endereçamento compartilhado permite que um único processador use quase toda a memória do computador. Portanto, um programa seqüencial em um cluster possui $1/n$ da memória disponível comparado com um programa seqüencial em um SMP.

A principal diferença entre os dois é o preço de aquisição para capacidade de computação equivalente para máquinas de grande escala. Como os multiprocessadores de grande escala possuem pequenos volumes, os custos extras de desenvolvimento de grandes máquinas precisam ser amortizados entre poucos sistemas, resultando em um custo mais alto para o consumidor. Uma vez que os mesmos switches vendidos em grande volume para pequenos sistemas podem ser usados para construir grandes redes para grandes clusters, os switches de rede local têm as mesmas vantagens de economia de escala que os computadores pequenos.

A limitação de memórias separadas para tamanho de programa torna-se uma vantagem na disponibilidade do sistema. Como um cluster consiste em computadores independentes conectados por meio de uma rede local, é muito mais fácil substituir uma máquina sem paralisar o sistema em um cluster do que em um SMP. Fundamentalmente, o endereçamento compartilhado significa que é difícil isolar um processador e substituí-lo sem um árduo trabalho por parte do sistema operacional. Já

que o software de cluster é uma camada que roda sobre o sistema operacional executado em cada computador, é muito mais fácil desconectar e substituir uma máquina defeituosa.

Como os clusters são construídos por meio de computadores inteiros e redes independentes e escaláveis, esse isolamento também facilita expandir o sistema sem paralisar a aplicação que roda sobre o cluster. A alta disponibilidade e a rápida e gradual expansibilidade tornam os clusters atraentes para provedores de serviços para a World Wide Web.

Em um lado da batalha, para combater a limitação de alta disponibilidade dos multiprocessadores, os projetistas de hardware e os desenvolvedores de sistemas operacionais estão procurando oferecer a capacidade de executar múltiplos sistemas operacionais em partes da máquina completa, de modo que um nó possa falhar ou ser atualizado sem desativar a máquina inteira.

No outro lado da batalha, já que tanto a administração do sistema quanto os limites de tamanho de memória são aproximadamente lineares no número de máquinas independentes, algumas pessoas estão reduzindo os problemas de cluster construindo clusters a partir de SMP de pequena escala. Por exemplo, um cluster de 32 processadores pode ser construído com oito SMPs de quatro processadores ou quatro SMPs de oito processadores. Esses clusters "híbridos" – algumas vezes chamados **constelações**, ou *clusters de memória compartilhada* – estão se tornando populares com aplicações que se importam com custo/desempenho, disponibilidade e expansibilidade. A Figura 9.1.1 mostra que cerca da metade dos clusters nos 500 melhores supercomputadores contém estações de trabalho de processador único e cerca de metade dos clusters contém servidores SMPs.

constelação Um cluster que usa SMP como o bloco de construção.

9.6 Topologias de rede

O Capítulo 8 examinou as redes locais comutadas, facilmente encontradas no mercado, que são a base dos clusters. Nesta seção, descreveremos as redes proprietárias usadas em multiprocessadores.

Os custos de rede incluem o número de chaves, o número de links em uma chave para conectar à rede, a largura (número de bits) por link e a extensão dos links quando a rede é mapeada para uma máquina física. Por exemplo, em uma máquina que escala entre dezenas e centenas de processadores, alguns links podem ser retângulos de metal dentro de um chip que possuem alguns milímetros de comprimento, e outros podem ser cabos que precisam percorrer vários metros de um rack até outro. O desempenho de rede também é multifacetado. Ele inclui a latência em uma rede não carregada para enviar e receber uma mensagem, a vazão em termos do número máximo de mensagens que podem ser transmitidas em um determinado período, atrasos causados por contenção para uma parte da rede e desempenho variável dependendo do padrão de comunicação. Outro requisito da rede pode ser a tolerância a falhas, já que sistemas muito grandes podem precisar operar na presença de componentes quebrados.

As redes normalmente são desenhadas como gráficos, com cada arco do gráfico representando um link da rede de comunicação. O nó processador-memória é mostrado como um quadrado preto, e a chave é mostrada como um círculo em cinza. Nesta seção, todos os links são *bidirecionais*; ou seja, as informações podem fluir em qualquer direção. Todas as redes consistem de *chaves* cujos links vão para os nós processador-memória e para outras chaves. A primeira melhoria sobre um barramento é uma rede que conecta uma seqüência de nós:

Essa topologia é chamada de *anel*. Como alguns nós não estão diretamente conectados, algumas mensagens precisarão saltar por nós intermediários até chegarem no destino final.

Ao contrário de um barramento, um anel é capaz de muitas transferências simultâneas. Como existem inúmeras topologias a escolher, são necessárias métricas de desempenho para distinguir esses projetos. Duas delas são populares. A primeira é a **largura de banda de rede** *total*, que é a largura de banda de cada link multiplicada pelo número de links. Isso representa o melhor caso. Para essa rede em anel com P processadores, a largura de banda de rede total seria P vezes a largura de banda de um link; a largura de banda de rede total de um barramento é exatamente a largura de banda desse barramento, ou duas vezes a largura de banda desse link.

largura de banda de rede Informalmente, a taxa de transferência de pico de uma rede. Pode se referir à velocidade de um único link ou à taxa de transferência coletiva de todos os links na rede.

Para equilibrar esse melhor caso, incluímos outra métrica mais próxima do pior caso: a *largura de banda de bisseção*. Isso é calculado dividindo-se a máquina em duas partes, cada uma com a metade dos nós. Depois, você soma a largura de banda dos links que interceptam essa linha divisória imaginária. A largura de banda de bisseção de um anel é duas vezes a largura de banda de link e é uma vez a largura de banda para o barramento. Se um único link for tão rápido quanto o barramento, o anel será apenas duas vezes mais rápido quanto um barramento no pior caso, mas será P vezes mais rápido no melhor caso.

Como algumas topologias de rede não são simétricas, a questão é onde desenhar a linha imaginária quando criar a bisseção da máquina. Essa é uma métrica de pior caso; portanto, a resposta é escolher a divisão que produz o desempenho de rede mais pessimista; ou seja, calcule todas as larguras de banda de bisseção possíveis e escolha a menor. Empregamos essa visão pessimista porque os programas paralelos normalmente são limitados pelo link mais fraco na cadeia de comunicação.

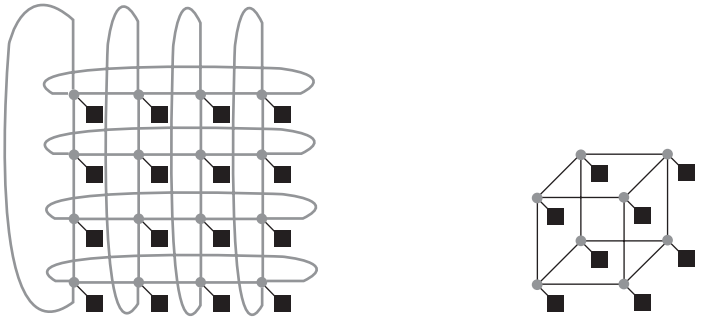
rede totalmente conectada Uma rede que conecta nós processador-memória fornecendo um link de comunicação dedicado entre cada nó.

No outro extremo de um anel está uma **rede totalmente conectada**, na qual cada processador possui um link bidirecional para todos os outros processadores. Para as redes totalmente conectadas, a largura de banda de rede total é $P \times (P - 1)/2$ e a largura de banda de bisseção é $(P/2)^2$.

A enorme melhoria no desempenho das redes totalmente conectadas é contrabalançada pelo enorme aumento do custo. Isso inspira os engenheiros a desenvolverem novas topologias que estejam entre o custo dos anéis e o desempenho das redes totalmente conectadas. A avaliação do sucesso depende, em grande parte, da natureza da comunicação no workload dos programas paralelos executados na máquina.

Seria difícil contar o número de topologias diferentes que têm sido discutidas nas publicações, mas o número que tem sido usado nos processadores paralelos comerciais é apenas um punhado. A Figura 9.6.1 ilustra duas topologias comuns. As máquinas reais geralmente acrescentam links extras a essas topologias simples para melhorar o desempenho e a confiabilidade.

Uma alternativa para colocar um processador em cada nó em uma rede é deixar apenas a chave em alguns desses nós. As chaves são menores que os nós processador-memória-chave e, portanto, po-



a. Grade ou malha bidimensional de 16 nós b. Árvore n-cubo de 8 nós ($8 = 2^3$, então, n = 3)

FIGURA 9.6.1 Topologias de rede que aparecem nos processadores paralelos comerciais. Os círculos representam chaves e os quadrados representam nós processador-memória. Mesmo que uma chave tenha muitos links, normalmente apenas um chega no processador. A topologia booleana de n-cubo é uma interconexão de n dimensões com 2^n nós, exigindo n links por chave (mais um para o processador) e, portanto, n nós vizinhos mais próximos. Essas topologias básicas freqüentemente são suplementadas com arcos extras para melhorar o desempenho e a confiabilidade.

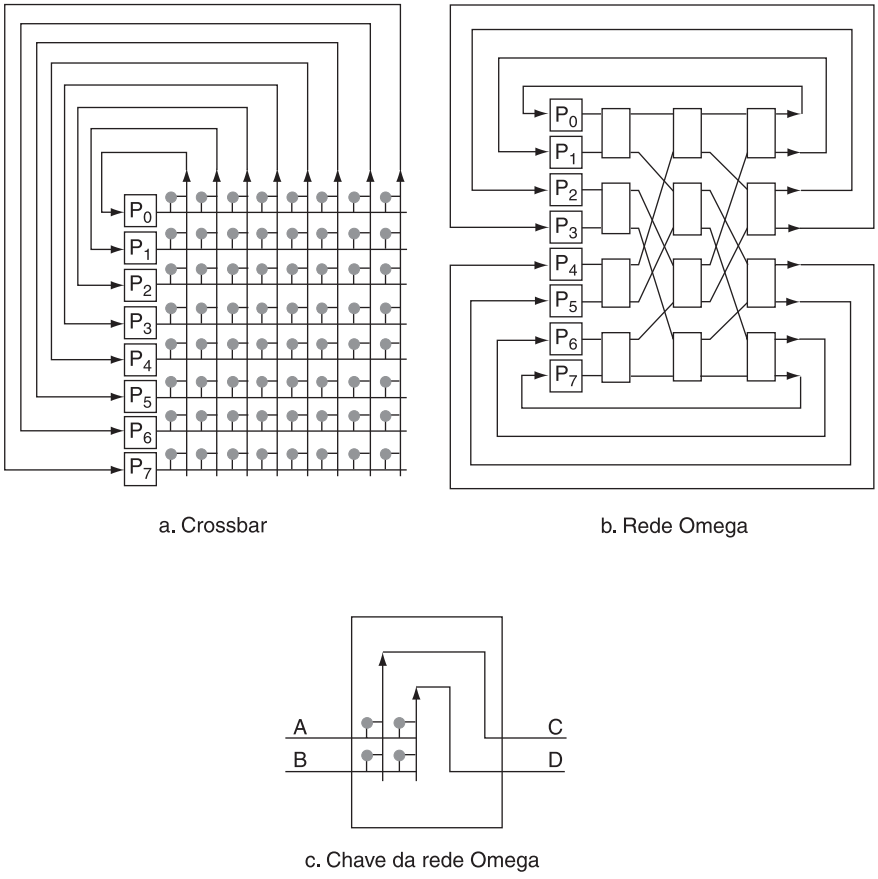


FIGURA 9.6.2 Topologias de rede multiestágio comuns para oito nós. As chaves nesses desenhos são mais simples do que nos desenhos anteriores porque os links são unidirecionais; os dados entram na parte inferior e saem no link direito. A chave em *c* pode passar A para C e B para D ou B para C e A para D. A crossbar usa n^2 chaves, onde n é o número de processadores, enquanto a rede Omega usa $n/2 \log_2 n$ grandes chaves, cada uma composta logicamente de quatro chaves menores. Nesse caso, a crossbar usa 64 chaves contra 12 chaves grandes, ou 48 chaves, na rede Omega. Entretanto, a crossbar pode suportar qualquer combinação de mensagens entre processadores, enquanto a rede Omega não pode.

redes multiestágio Uma rede que fornece uma pequena chave em cada nó.

rede totalmente conectada Uma rede que conecta nós processador-memória fornecendo um link de comunicação dedicado entre cada nó.

rede crossbar Uma rede que permite que qualquer nó se comunique com qualquer outro nó em uma passagem pela rede.

dem ser agrupadas mais densamente, diminuindo, assim, a distância e aumentando o desempenho. Essas redes são chamadas de **redes multiestágio** para refletir as múltiplas etapas que uma mensagem precisa percorrer. Os tipos de redes multiestágio são tão numerosos quanto as redes de estágio único; a Figura 9.6.2 ilustra duas organizações multiestágio comuns. Uma **rede totalmente conectada** ou uma **rede crossbar** permite que qualquer nó se comunique com qualquer outro nó em uma passagem pela rede. Uma *rede Omega* usa menos hardware do que a rede crossbar ($2n \log_2 n$ contra n^2 chaves), mas pode ocorrer contenção entre mensagens, dependendo do padrão de comunicação. Por exemplo, a rede Omega na Figura 9.6.2 não pode enviar uma mensagem de P0 para P6 ao mesmo tempo em que envia uma mensagem de P1 para P7.

Implementando topologias de rede

Essa análise simples de todas as redes nesta seção ignora importantes considerações práticas na construção de uma rede. A distância de cada link afeta o custo da comunicação em uma alta velocidade de clock – em geral, quanto mais longa a distância, mais caro é funcionar a uma alta velocidade de clock. Distâncias mais curtas também facilitam a utilização de mais fios por link, já que a energia necessária para conduzir muitos fios em um chip é menor se os fios forem curtos. Fios mais curtos também são mais baratos do que fios mais longos. Uma última limitação prática é que os desenhos

bidimensionais precisam ser mapeados para chips e placas que são essencialmente meios bidimensionais. A conclusão é que as topologias que parecem elegantes quando desenhadas no quadro-negro podem parecer desajeitadas quando construídas com chips, cabos, placas e caixas.

9.7 Multiprocessadores no interior de um chip e multithreading

Uma alternativa aos multiprocessadores que compartilham uma interconexão é trazer os processadores para dentro do chip. Nesses projetos, os processadores normalmente compartilham algumas das caches e a interface de memória externa. Claramente, as latências associadas à comunicação chip a chip desaparecem quando tudo está no mesmo chip. A questão é: "Qual é o impacto dos processadores na hierarquia de memória?". Se eles estiverem executando o mesmo código, como um banco de dados, então, os processadores pelo menos amortizam os acessos de instrução. As estruturas de dados compartilhadas também são um problema muito menor quando as caches são compartilhadas. Várias empresas anunciaram microprocessadores com múltiplos núcleos por chip.

A idéia de aumentar a utilização de recursos em um chip por meio da execução paralela de threads encontrou outra implementação. O *multithreading* permite que várias threads compartilhem as unidades funcionais de um único processador de um modo sobreposto. Para permitir esse compartilhamento, o processador precisa duplicar o estado independente de cada thread. Por exemplo, uma cópia separada do banco de registradores, um PC separado e uma tabela de páginas separada são necessários para cada thread. A memória em si pode ser compartilhada por meio de mecanismos de memória virtual, que já suportam multiprogramação. Além disso, o hardware precisa suportar a capacidade de mudar para uma thread diferente com relativa rapidez; em especial, uma troca de thread deve ser muito mais eficiente do que uma troca de processo, que normalmente exige centenas a milhares de ciclos de processador.

Existem dois métodos principais de multithreading. O *multithreading fine-grained* comuta entre threads a cada instrução, resultando em execução intercalada de várias threads. Essa intercalação é feita de forma circular, saltando quaisquer threads que estejam suspensas nesse momento. Para tornar o multithreading fine-grained prático, o processador precisa ser capaz de trocar threads a cada ciclo de clock. Uma importante vantagem do multithreading fine-grained é que ele pode ocultar as perdas de vazão que surgem dos stalls curtos e longos, já que as instruções de outras threads podem ser executadas quando uma thread é suspensa. A principal desvantagem do multithreading fine-grained é que ele torna mais lenta a execução das threads individuais, já que uma thread que está pronta para ser executada sem stalls será atrasada por instruções de outras threads.

O *multithreading coarse-grained* foi criado como uma alternativa para o multithreading fine-grained. Esse método de multithreading comuta threads apenas em stalls onerosos, como as falhas de cache de nível 2. Essa mudança reduz a necessidade de tornar a comutação de thread essencialmente gratuita e tem muito menos chance de tornar mais lenta a execução de uma thread individual, visto que só serão despachadas instruções de outras threads quando uma thread encontrar um stall oneroso. Entretanto, o multithreading coarse-grained sofre de uma grande desvantagem: Ele é limitado em sua capacidade de sanar perdas de vazão, especialmente de stalls mais curtos. Essa limitação surge dos custos de inicialização de pipeline do multithreading coarse-grained. Como uma CPU com multithreading coarse-grained despacha instruções por meio de uma única thread, quando um stall ocorre, o pipeline precisa ser esvaziado ou congelado. A nova thread que começa a ser executada após o stall precisa preencher o pipeline antes que as instruções consigam ser concluídas. Devido a esse overhead de inicialização, o multithreading coarse-grained é muito mais útil para reduzir a penalidade dos stalls de alto custo, onde a reposição de pipeline é insignificante comparada com o tempo de stall.

O *simultaneous multithreading* (SMT) é uma variação de multithreading que usa os recursos de um processador de despacho múltiplo escalonado dinamicamente para explorar paralelismo em nível de thread ao mesmo tempo em que explora o paralelismo em nível de instrução. O princípio mais importante que motiva o SMT é que os processadores de despacho múltiplo modernos normalmente possuem mais paralelismo de unidade funcional do que uma única thread efetivamente pode usar. Além disso, com a renomeação de registradores e o escalonamento dinâmico, diversas instruções de threads independentes podem ser despachadas inconscientemente das dependências entre elas; a resolução das dependências pode ser tratada pela capacidade de escalonamento dinâmico.

A Figura 9.7.1 ilustra conceitualmente as diferenças na capacidade de um processador de explorar recursos superescalares para as configurações de processador a seguir. A parte superior mostra como quatro threads seriam executadas de forma independente em um superescalar sem suporte a multithreading. A parte inferior mostra como as quatro threads poderiam ser combinadas para serem executadas no processador de maneira mais eficiente usando três opções de multithreading:

- Um superescalar com multithreading coarse-grained
- Um superescalar com multithreading fine-grained
- Um superescalar com simultaneous multithreading

No superescalar sem suporte a multithreading, o uso dos slots de despacho é limitado por uma falta de paralelismo em nível de instrução. Além disso, um importante stall, como uma falha de cache de instruções, pode deixar o processador inteiro ocioso.

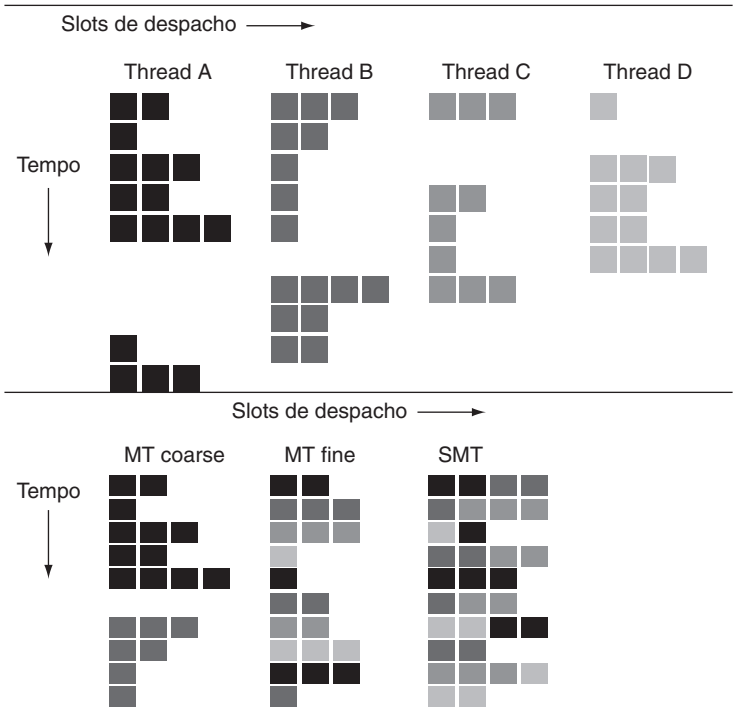


FIGURA 9.7.1 Como quatro threads usam os slots de despacho de um processador superescalar em diferentes métodos. As quatro threads no alto mostram como cada uma seria executada em um processador superescalar padrão sem suporte a multithreading. Os três exemplos embaixo mostram como elas seriam executadas juntas em três opções de multithreading. A dimensão horizontal representa a capacidade de despacho de instrução em cada ciclo de clock. A dimensão vertical representa uma seqüência dos ciclos de clock. Uma caixa vazia (branco) indica que o slot de despacho correspondente está vago nesse ciclo de clock. Os tons de cinza e preto correspondem a quatro threads diferentes nos processadores multithreading. Os efeitos de inicialização de pipeline adicionais para MT coarse, que não estão ilustrados nessa figura, levariam a mais perda na vazão para MT coarse.

No superescalar com multithreading coarse-grained, os longos stalls são parcialmente ocultados pela comutação para outra thread que usa os recursos do processador.

Embora isso reduza o número de ciclos de clock completamente ociosos, dentro de cada ciclo de clock, as limitações do paralelismo em nível de instrução ainda produzem ciclos ociosos. Além disso, em um processador com multithreading coarse-grained, como a comutação de thread ocorre apenas quando há um stall e a nova thread tem um período de inicialização, é provável que ainda restem ciclos ociosos.

No caso fine-grained, a intercalação dos threads elimina os slots completamente vazios. No entanto, como apenas uma thread despacha instruções em um determinado ciclo de clock, as limitações do paralelismo em nível de instrução ainda geram um número significativo de slots ociosos dentro dos ciclos de clock individuais.

No caso SMT, o paralelismo em nível de thread (TLP) e o paralelismo em nível de instrução (ILP) são explorados simultaneamente, com múltiplas threads usando os slots de despacho em um único ciclo de clock. O ideal é que o uso de slots de despacho seja limitado por desequilíbrios nas necessidades e na disponibilidade de recursos entre múltiplas threads. Na prática, outros fatores – como a quantidade de threads ativas consideradas, as limitações finitas dos buffers, a capacidade de buscar instruções suficientes das múltiplas threads e as limitações práticas de que combinações de instruções podem ser despachadas de uma thread e de várias threads – também podem restringir o número de slots usados. Embora a Figura 9.7.1 simplifique bastante a operação real desses processadores, ela ilustra as potenciais vantagens de desempenho do multithreading em geral e do SMT em particular.

Como mencionado anteriormente, o simultaneous multithreading usa o princípio de que um processador com escalonamento dinâmico já tem muitos dos mecanismos de hardware necessários para suportar a exploração integrada do TLP por meio do multithreading. Em particular, os processadores superescalares com escalonamento dinâmico têm um grande conjunto de registradores que podem ser usados para conter os conjuntos de registradores de threads independentes (considerando que tabelas de renomeação separadas são mantidas para cada thread). Como a renomeação de registradores fornece identificadores de registrador únicos, as instruções de múltiplas threads podem ser misturadas no caminho de dados sem que origens e destinos sejam confundidos entre as threads. Essa observação nos leva ao princípio de que o multithreading pode ser construído sobre um processador com execução fora de ordem incluindo uma tabela de renomeação por thread, mantendo PCs separados e fornecendo a capacidade de término de instruções de múltiplas threads. Existem complicações para tratar o término de instruções, já que gostaríamos que instruções de threads independentes fossem capazes de terminar independentemente. O término independente de instruções de threads separadas pode ser suportado mantendo logicamente um buffer de reordenação separado para cada thread.

Existe uma variedade de outros desafios de projeto para um processador SMT, incluindo os seguintes:

- Lidar com um banco de registradores maior, necessário para conter vários contextos
- Manter o overhead baixo no ciclo de clock, particularmente em fases críticas como o despacho de instrução, no qual mais instruções candidatas precisam ser consideradas, e no término da instrução, no qual pode ser complicado escolher as instruções a serem finalizadas
- Garantir que os conflitos de cache gerados pela execução simultânea das múltiplas threads não causem redução de desempenho significativa

Analisando esses problemas, duas observações são importantes. Primeiro, em muitos casos, o overhead de desempenho potencial devido ao multithreading é pequeno e escolhas simples funcionam bastante bem. Segundo, a eficiência dos superescalares é tão baixa que existe espaço para uma significativa melhoria, mesmo ao custo de algum overhead. O SMT parece ser o método mais promissor de conseguir essa melhoria na vazão.

A Intel chama seu suporte de SMT no Pentium 4 de *Hyper-Threading*. Dobrando o estado arquitetural do IA-32, ele suporta apenas duas threads, que compartilham todas as caches e unidades funcionais.

9.8 Vida real: o cluster de PCs do Google

Os mecanismos de busca possuem uma grande necessidade de confiabilidade, já que são usados por pessoas em qualquer hora do dia e de qualquer lugar do mundo. O Google, essencialmente, precisa estar sempre disponível.

Como um mecanismo de busca costuma interagir com uma pessoa, sua latência não deve exceder a paciência dos seus usuários. A meta do Google é que nenhuma busca leve mais de 0,5 segundo, já considerando os atrasos da rede. A largura de banda também é vital. O Google serve uma *média* de aproximadamente 1.000 consultas por segundo e já buscou e indexou mais de 3 bilhões de páginas.

Além disso, um mecanismo de busca precisa pesquisar a Web regularmente para ter informações atualizadas. O Google examina toda a Web e atualiza seu índice a cada quatro semanas, de modo que toda página Web é visitada uma vez por mês. O Google também mantém uma cópia local do texto da maioria das páginas para que possa fornecer o fragmento do texto bem como uma cópia em cache da página.

Para fazer frente a essa demanda, o Google usa mais de 6.000 processadores e 12.000 discos, o que lhe confere um total de aproximadamente 1 petabyte de armazenamento em disco.

Em vez de conseguir disponibilidade usando armazenamento RAID, o Google se baseia em sites redundantes, cada um com milhares de discos e processadores: dois sites no Vale do Silício e dois na Virgínia. O índice de busca, que é um pequeno número de terabytes, além do depósito de páginas em cache, que está na ordem no mesmo tamanho, são duplicados ao longo do sites. Portanto, se um site falhar, ainda existem dois outros sites que podem sustentar o serviço. Além disso, o índice e o depósito são duplicados dentro de um site para ajudar a compartilhar o workload e a continuar a fornecer serviço dentro de um site mesmo se os componentes falharem.

Cada site está conectado à Internet por meio de links OC48 (2488 Mbit/s) do site de colocação. Para se precaver de falhas do link de colocação, existe um link OC12 separado conectando um par de sites de modo que, em uma emergência, ambos os sites possam usar o link Internet de um dos sites. O link externo é improvável de falhar em ambos os sites, uma vez que diferentes provedores de rede fornecem linhas OC48.

A Figura 9.8.1 mostra a planta baixa de um site típico. O link OC48 é conectado a dois switches Ethernet de 128 × 128 por meio de um grande switch. Observe que esse link também está conectado ao restante dos servidores no site. Esses dois switches são redundantes de modo que uma falha de switch não desconecte o site. Também há um link OC12 dos switches Ethernet de 128 × 128 para o site irmão, para emergências. Cada switch pode se conectar a 128 linhas Ethernet de 1 Gbit/seg. Racks de PCs, cada um com 4 interfaces Ethernet de 1 Gbit/s, são conectados aos dois switches Ethernet de 128 × 128. Portanto, um único site pode suportar 2 × 128/4 ou 64 racks de PCs.

A Figura 9.8.2 mostra o rack de PCs do Google. O Google usa PCs que são apenas 1 unidade de rack VME. Para conectar esses PCs aos switches Ethernet de 128 × 128, ele usa um pequeno switch Ethernet. Ele possui 4 RU de altura, deixando espaço no rack para 40 PCs. Esse switch possui interfaces de rede modulares, organizadas como *lâminas* removíveis. Cada lâmina pode conter 8 interfaces Ethernet de 100Mbit/s ou uma única interface de rede. Portanto, 5 lâminas são usadas para conectar cabos Cat5 de 100Mbit/s a cada um dos 40 PCs no rack, e 2 lâminas são usadas para conectar cabos de cobre de 1Gbit/s a dois switches Ethernet de 128 × 128.

Para compactar ainda mais PCs, o Google escolheu um rack que oferece a mesma configuração *na frente e atrás*, resultando em 80 PCs e dois switches por rack, como mostra a Figura 9.8.2. Esse sistema possui um espaço de aproximadamente 91,44cm no centro entre as colunas de PCs para a saída do ar quente, que é retirada da "chaminé" por meio de ventoinhas de exaustão no alto do rack.

O PC propriamente dito é bastante comum: duas unidades ATA/IDE, 256MB de SDRAM, um modesto microprocessador Intel, uma placa-mãe de PC, uma fonte de alimentação e algumas ventoinhas. Cada PC executa o sistema operacional Linux. Para tirar o máximo proveito do investimento, a cada 2 a 3 meses o Google aumenta a capacidade das unidades ou a velocidade do processador. Por-

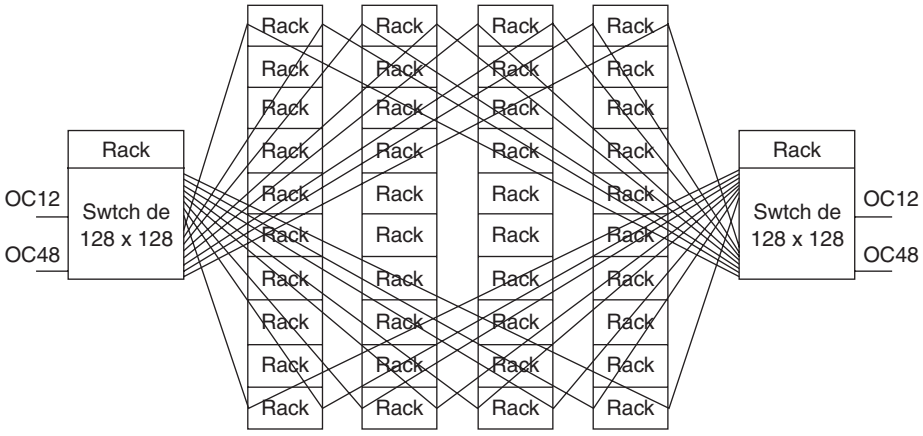


FIGURA 9.8.1 Planta baixa de um cluster do Google (visão aérea). Existem 40 racks, cada um conectado por meio de links Ethernet de cobre de 4Gbit a dois switches redundantes de 128 × 128. A Figura 9.8.2 mostra que um rack contém 80 PCs, de modo que essa instalação contém aproximadamente 3.200 PCs. (Para maior clareza, os links são mostrados apenas para o rack superior e inferior de cada fila.) Esses racks estão em um piso elevado para que os cabos possam estar ocultos e protegidos. Cada switch Ethernet de 128 × 128, por sua vez, está conectado à rede do site de colocação por meio de um link OC48 (2.4 Gbit) à Internet. Existem dois switches Ethernet de 128 × 128, de modo que o cluster ainda permaneça conectado mesmo se um switch falhar. Também há um link OC12 (622 Mbit) separado próximo ao site de colocação para o caso de a rede OC48 do site de colocação falhar; ele ainda serve tráfego pela OC12 à rede do outro site. Cada switch Ethernet de 128 × 128 pode manipular 128 linhas Ethernet de 1Gbit e cada rack possui 2 linhas Ethernet de 1Gbit por switch, de modo que o número máximo de racks para o site é 64. Os dois racks próximos aos switches Ethernet de 128 × 128 contêm alguns PCs para atuarem como front-ends e ajudarem em tarefas como serviço HTML, balanceamento de carga, monitoramento e no-break para manter o switch e os fronts ativos no caso de uma curta interrupção de energia. Parece que uma instalação que possui motores a diesel redundantes para fornecer energia independente para todo o site tornaria o no-break redundante. Uma pesquisa de usuários dos centros de dados sugere que falhas de energia ainda acontecem anualmente.

tanto, o site de 40 racks mostrado na Figura 9.8.1 foi preenchido com microprocessadores que variaram de um Celeron 533MHz a um Pentium III 800MHz, discos que variaram em capacidade entre 40 e 80GB e em velocidade entre 5.400 e 7.200RPM, e velocidade de barramento de memória que era de 100 ou 133MHz.

Desempenho

Cada site de colocação se conecta à Internet por meio de links OC48 (2488 Mbit/s), que é compartilhado pelo Google e outros provedores de serviço de Internet. Se uma resposta típica a uma consulta possui, digamos, 4.000 bytes, e se o Google serve 100 milhões de consultas por dia, então, a demanda de largura de banda média é

$$\frac{100.000.000 \text{ consultas/dia } 4.000 \text{ bytes/consulta} \times 8 \text{ bits/byte}}{24 \times 60 \times 60 \text{s/dia}} = \frac{3.200.000 \text{Mbits}}{86.400 \text{s}} \approx 37 \text{Mbits/s}$$

que é apenas 1,5% da velocidade de link de cada site. Mesmo se multiplicarmos por um fator de 4 para considerar demanda de pico *versus* média de requisições e respostas, o Google precisa de pouco dessa largura de banda.

Examinar a Web e atualizar os sites exige muito mais largura de banda do que servir as consultas. Vamos estimar alguns parâmetros para colocar as coisas em perspectiva. Considere que sejam necessários 7 dias para examinar 3 bilhões de páginas:

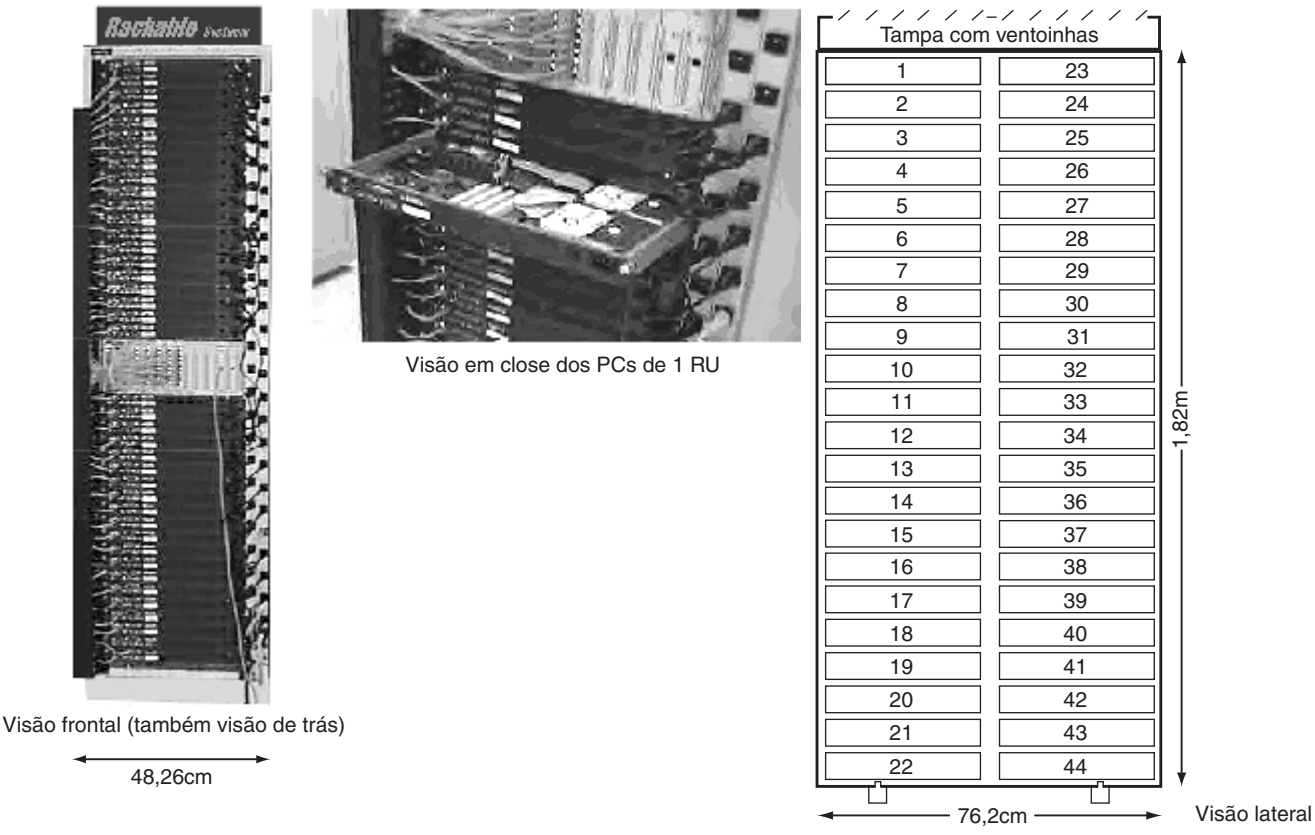


FIGURA 9.8.2 Visão frontal, visão lateral e close de um rack de PCs usado pelo Google. A fotografia à esquerda mostra o pequeno switch Ethernet no centro, com 20 PCs acima e 20 PCs abaixo no rack Rackable Systems. Cada PC está conectado ao switch no centro por meio de um cabo Cat5 no lado esquerdo, com Ethernet de 100Mbit. Cada "lâmina" do switch pode conter oito interfaces Ethernet de 100Mbit ou uma interface de 1Gbit. Também há dois links Ethernet de 1 Gbit saindo do switch à direita. Portanto, cada PC tem apenas dois cabos: um Ethernet e um cabo de força. A extremidade direita da foto mostra uma barra de energia, com cada um dos 40 PCs e o switch conectados nela. Cada PC possui 1 unidade de rack (RU) VME de altura. O switch no meio possui 4 RU de altura. A foto ao centro é um close de um rack, mostrando o conteúdo de um PC de 1 RU. Essa unidade contém duas unidades IDE de 5.400RPM à direita da caixa, 256MB de SDRAM de 100MHz, uma placa-mãe de PC, uma única fonte de alimentação e um microprocessador Intel. Cada PC executa versões 2.2.16 ou 2.2.17 do kernel Linux em uma distribuição Red Hat ligeiramente modificada. Você pode ver os cabos Ethernet à esquerda, os cabos de força à direita e cabos Ethernet conectados ao switch no alto da figura. Em dezembro de 2000, o custo das partes não separadas era aproximadamente US$500 para os dois discos, US$200 para o microprocessador, US$100 para a placa-mãe e US$100 para a DRAM. Incluindo o gabinete, a fonte de alimentação, as ventoinhas, o cabeamento etc., um PC montado pode custar de US$1.300 a US$1.700. O desenho à direita mostra que os PCs são mantidos em duas colunas, frontal e traseira, de modo que um único rack contém 80 PCs e dois switches. A potência típica por PC é aproximadamente 55 watts e cerca de 70 watts por switch, de modo que um rack consome cerca de 4.500 watts. O calor é exaurido para uma abertura entre as duas colunas e o ar quente é retirado pelo alto usando ventoinhas. (O desenho mostra 22 PCs por lado, cada um com 2 RU de altura, em vez da configuração do Google de 40 PCs de 1 RU mais um switch por lado.)

$$\frac{3.000.000.000 \text{ páginas} \times 4000\text{bytes/página} \times 8\,\text{bits/byte}}{24 \times 60 \times 60\text{s/dia} \times 7 \text{ dias}} = \frac{96.000.000\text{Mbits}}{604.800\text{s}} \approx 159\text{Mbits/s}$$

Esses dados são coletados em um único site, mas o índice e o depósito final de vários terabytes precisam, então, ser duplicados nos outros dois sites. Se considerarmos que temos 7 dias para duplicar os dados e que estamos enviando, digamos, 15TB de um site para dois sites, então, a demanda de largura de banda média é

$$\frac{15.000.000\text{MB/8 bits/byte}}{2 \times 24 \times 60 \times 60\text{s/dia} \times 7 \text{ dias}} = \frac{240.000.000\text{Mbits}}{604.800\text{s}} \approx 396\text{Mbits/s}$$

Portanto, a largura de banda máquina-pessoa é relativamente insignificante, com a largura de banda real sendo de máquina para máquina. Essa ainda é uma pequena fração dos 2.488Mbits/s disponíveis.

O tempo de vôo para mensagens por meio dos Estados Unidos é de aproximadamente 0,1 segundo; então, é importante para a Europa ser servida por sites da Virgínia e para a Califórnia ser servida por sites do Vale do Silício. Para tentar atingir a meta de 0,5 segundo de latência, o software do Google normalmente estima de onde a busca se origina a fim de reduzir os atrasos de tempo de vôo.

Custo

Visto que o bloco básico de construção do cluster do Google é um PC, o custo de capital de um site normalmente é uma função do custo de um PC. Em vez de comprar o microprocessador mais recente, o Google procura o de melhor custo-desempenho. Estimamos que o custo de PC era de US$1.300 a US$1.700.

Os switches custam cerca de US$1.500 para o switch Ethernet pequeno e cerca de US$100.000 cada uma para os switches Ethernet de 128 × 128. Se os racks custam aproximadamente US$1.000 a US$2.000 cada um, o custo de capital total de um site de 40 racks é de aproximadamente US$4,5 milhões a US$6 milhões. Incluindo 3.200 microprocessadores e 0,8TB de DRAM, o armazenamento em disco custa cerca de US$10.000 a US$15.000 por terabyte. Se eles tivessem comprado servidores e arrays de disco padrão nesse espaço de tempo, seu custo teria sido 5 a 10 vezes mais alto.

O rack do Google com 80 PCs, com cada PC operando em aproximadamente 55W, usa 4.500W em 0,92 metro quadrado. Ele é consideravelmente mais alto do que os 1.000W por rack esperados pelos sites de colocação. Cada rack do Google usa 60amps. Como mencionado anteriormente, reduzir o consumo por PC é uma grande oportunidade para o futuro desses clusters, especialmente porque o custo por kilowatt-hora está aumentando e o custo por Mbit/s está diminuindo.

Confiabilidade

A maior origem de falhas de componente no cluster de PCs do Google é o software. Em um dia médio, cerca de 20 máquinas serão reiniciadas e isso normalmente resolve o problema. Para reduzir o número de cabos por PC, bem como o custo, o Google não tem a capacidade de reiniciar remotamente uma máquina. O software pára de enviar trabalho a uma máquina quando observa um comportamento incomum, o operador chama o site de colocação e informa o local da máquina que precisa ser reiniciada, e uma pessoa no site encontra o rótulo e pressiona o botão no painel frontal. Ocasionalmente, a pessoa pressiona o botão errado por engano ou por rotulação incorreta no lado de fora da caixa.

O próximo problema de confiabilidade de componente é o hardware, que apresenta cerca de 1/10 das falhas de software. Normalmente, cerca de 2% a 3% dos PCs precisam ser substituídos por ano, sendo que as falhas devido a discos e DRAM respondem por 95% dessas falhas. Os 5% restantes são devido a problemas com a placa-mãe, fonte de alimentação, conectores etc. os microprocessadores em si nunca parecem falhar.

As falhas de DRAM talvez sejam um terço das falhas de componentes de hardware. O Google vê erros tanto de mudança de bits dentro da DRAM quanto de transferência de bits pelo barramento de 100 a 133MHz. Como não havia qualquer proteção de ECC disponível para os chip-sets da placa-mãe do desktop PC no momento, ela não era usada. A DRAM é determinada como sendo o problema quando o Linux não pode ser instalado com um checksum correto até que a DRAM seja substituída. O Google planeja usar ECC para corrigir algumas falhas, mas, principalmente, para facilitar saber quando as DRAMs falham. O custo extra do ECC é insignificante devido à grande flutuação nos preços da DRAM; procedimentos de compra cuidadosos são mais importantes do que o fato de a DIMM ter ECC.

Os discos são a última causa das falhas do PC. Além das falhas comuns que resultam em uma mensagem para o log de erro no console, em números quase iguais, esses discos ocasionalmente re-

sultarão em uma *falha de desempenho*, sem qualquer mensagem de erro para o log. Em vez de gerar larguras de banda de leitura normais em 28MB/s, os discos repentinamente cairão para 4MB/s ou mesmo 0,8MB/s. Como os discos estão sob garantia por cinco anos, o Google devolve para o fabricante os discos com falhas operacionais ou de desempenho a fim de obter substituições. Portanto, não tem havido qualquer exploração do motivo para as anomalias de disco.

Quando um PC apresenta problemas, ele é reconfigurado fora do sistema e, aproximadamente uma vez por semana, uma pessoa remove os PCs defeituosos. Eles normalmente são reparados e, então, reinseridos no rack.

Com relação aos switches, em um período de mais de dois anos, talvez 200 dos pequenos switches Ethernet foram instalados e 2 ou 3 falharam. Nenhum dos seis switches Ethernet de 128 × 128 falhou no campo, embora alguns tenham tido problemas na distribuição. Esses switches possuem um desenho baseado em lâminas com 16 lâminas por switch e 2 ou 3 das lâminas falharam.

O último aspecto é a confiabilidade da colocação. Muitos provedores de serviço de Internet experimentam uma interrupção de energia por ano que afeta o site inteiro ou uma fração de um site. Em média, também há uma interrupção de rede de modo que o site inteiro seja desconectado da Internet. Essas interrupções podem durar horas.

O Google acomoda a falha de segurança da colocação tendo diversos sites com diferentes provedores de rede, além de linhas alugadas entre pares de sites para emergências. As faltas de energia, as interrupções de rede etc. não afetam a disponibilidade do serviço Google. O Google não teve sequer uma interrupção desde que a empresa tinha alguns meses de atividade.

Número 9: Cite o desempenho em termos de utilização do processador, speedups paralelos ou MFLOPS por dólar.

David H. Bailey, "Twelve ways to fool the masses when giving performance results on parallel supercomputers", *Supercomputing Review*, 1991

9.9 Falácias e armadilhas

Os muitos ataques ao processamento paralelo revelaram inúmeras falácias e armadilhas. Veremos três delas aqui.

Armadilha: medir o desempenho dos processadores paralelos por speedup linear versus *tempo de execução.*

Os gráficos "tiro de morteiro" – representação do desempenho comparado com o número de processadores, mostrando speedup linear, um platô e, depois, uma queda – foram usados por muito tempo para julgar o sucesso dos processadores paralelos. Embora a escalabilidade seja um aspecto de um programa paralelo, ela é uma medida indireta do desempenho. A primeira pergunta a ser feita se refere à capacidade dos processadores dimensionados: Um programa que melhora linearmente o desempenho para igualar a 100 processadores Intel 80386 pode ser mais lento do que a versão seqüencial em um único computador desktop Pentium 4.

Medir resultados usando speedup linear comparado com o tempo de execução pode iludir o programador bem como aqueles ouvindo as alegações de desempenho do programador. Muitos programas com fraco speedup são mais rápidos do que programas que mostram excelente speedup conforme aumenta o número de processadores.

Comparar tempos de execução é aconselhável apenas se você estiver comparando os melhores algoritmos em cada máquina. (É claro que você não pode subtrair tempo para processadores ociosos quando avaliar um processador paralelo; portanto, o tempo de CPU é uma métrica imprópria para processadores paralelos.) Comparar o código idêntico em duas máquinas pode parecer correto, mas não é; o programa paralelo pode ser mais lento em um uniprocessador do que em uma versão seqüencial. Algumas vezes, desenvolver um programa paralelo levará a melhorias algorítmicas; portanto, comparar o programa seqüencial anteriormente mais conhecido com o código paralelo – o que parece correto – compara algoritmos impróprios. Para refletir esse problema, os termos *speedup relativo* (mesmo programa) e *speedup real* (melhores programas) algumas vezes são usados.

Falácia: a Lei de Amdahl não se aplica aos computadores paralelos.

Em 1987, o diretor de uma organização de pesquisa afirmou que a Lei de Amdahl tinha sido quebrada por uma máquina de multiprocessador. Para tentar entender a base dos relatórios da mídia, vejamos a citação que nos trouxe a Lei de Amdahl [1967, p. 483]:

> *Uma conclusão bastante óbvia que pode ser tirada nesse momento é que o esforço despendido em conseguir altas velocidades de processamento paralelo é desperdiçado se não for acompanhado de conquistas de mesmas proporções nas velocidades de processamento seqüencial.*

Essa afirmação ainda deve ser verdadeira; a parte ignorada do programa deve limitar o desempenho. Uma interpretação da lei leva ao seguinte princípio: partes de cada programa precisam ser seqüenciais e, portanto, precisa haver um limite superior lucrativo para o número de processadores – digamos, 100. Mostrando speedup linear com 1.000 processadores, esse princípio se torna falso e, então, a Lei de Amdahl foi quebrada.

O método dos pesquisadores foi mudar a entrada para o benchmark: em vez de ir 1.000 vezes mais rápido, eles calcularam 1.000 vezes mais trabalho em tempo comparável. Para o algoritmo deles, a parte seqüencial do programa era constante, independente do tamanho da entrada, e o restante era totalmente paralelo – daí, speedup linear com 1.000 processadores. Simplesmente escalar o tamanho das aplicações, sem também escalar a precisão do ponto flutuante, o número de iterações, as necessidades de E/S e a maneira como as aplicações lidam com erros pode ser uma atitude ingênua. Muitas aplicações não calcularão o resultado correto se o tamanho do problema for inadvertidamente aumentado.

Não vemos razão para que a Lei de Amdahl não se aplique aos processadores paralelos. O que essa pesquisa salienta é a importância de ter benchmarks que possam crescer o suficiente para demonstrar o desempenho de processadores paralelos de grande escala.

Falácia: o desempenho de pico segue o desempenho observado.

Uma definição de desempenho de pico é "o desempenho que uma máquina seguramente não irá exceder". Infelizmente, a indústria de supercomputadores tem usado essa métrica no marketing e sua falácia está sendo extrapolada com as máquinas paralelas. Os marqueteiros do setor não só estão usando o quase inatingível desempenho de pico de um nó de uniprocessador (veja a Figura 9.9.1), mas também eles o estão multiplicando pelo número total de processadores, considerando um speedup perfeito! A Lei de Amdahl sugere como é difícil alcançar qualquer um dos picos; multiplicar um pelo outro também multiplica os erros. A Figura 9.9.2 compara o pico com o desempenho sustentado em um benchmark; o IBM SP2 de 64 processadores atinge apenas 7% do desempenho de pico. Claramente, o desempenho de pico nem sempre segue o desempenho observado.

Máquina	Avaliação de Pico em MFLOPS	Média harmônica dos benchmarks Perfect Club em MFLOPS	Porcentagem de pico em MFLOPS
Cray X-MP/416	940	14,8	1%
IBM 3090-600S	800	08,3	1%
NEC SX/2	1300	16,6	1%

FIGURA 9.9.1 Desempenho de pico e média harmônica de desempenho real para os 12 benchmarks Perfect Club. Os resultados são para os programas executados sem modificação. Quando ajustados manualmente, o desempenho das três máquinas vai para 24,4, 11,3 e 18,3 MFLOPS, respectivamente. Isso ainda é 2% ou menos do desempenho de pico.

	Cray YMP (8 processadores)		IBM SP2 (64 processadores)	
	MFLOPS	% do pico	MFLOPS	% do pico
Pico	2.666	100%	14.636	100%
3D FFT PDE	1.795	67%	1.093	7%

FIGURA 9.9.2 Desempenho de pico *versus* desempenho observado para o Cray YMP e o IBM RS/6000 SP2.

Essas alegações de desempenho podem confundir o fabricante tanto quanto o usuário da máquina. O perigo é que o fabricante desenvolva bibliotecas de software com sucesso julgado como porcentagem do desempenho de pico medido em megaflops em vez de levar menos tempo, ou que seja incluído hardware que aumente o desempenho de pico do nó, mas seja difícil de usar.

9.10 Comentários finais

Por mais de uma década os analistas anunciam que a organização de um único computador alcançou seus limites e que avanços verdadeiramente significantes só podem ser feitos pela interconexão de uma multiplicidade de computadores de tal modo que permita solução cooperativa... Demonstrou-se a continuada validade do método de processador único...

Gene Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities", *Spring Joint Computer Conference*, 1967

O sonho de construir computadores apenas agregando processadores existe desde os primeiros dias da computação. No entanto, o progresso na construção e no uso de processadores paralelos eficientes tem sido lento. Essa velocidade de progresso foi limitada pelos difíceis problemas de software bem como por um longo processo de evolução da arquitetura dos multiprocessadores para melhorar a usabilidade e a eficiência. Discutimos muitos dos problemas de software neste capítulo, incluindo a dificuldade de escrever programas que obtêm bom speedup devido à Lei de Amdahl. A grande variedade de métodos arquitetônicos diferentes e o sucesso limitado e a vida curta de muitas arquiteturas até agora se juntam às dificuldades de software. Abordaremos a história do desenvolvimento desses multiprocessadores na Seção 9.11.

Apesar desse longo e sinuoso passado, o progresso nos últimos 20 anos nos dá razões para sermos otimistas quanto ao futuro do processamento paralelo e dos multiprocessadores. Esse otimismo é baseado em diversas observações sobre esse progresso e as diretrizes de tecnologia de médio prazo:

- Agora, é amplamente aceito que a maneira mais eficaz de construir um computador que ofereça mais desempenho do que o obtido com um microprocessador de chip único é construindo um multiprocessador ou um cluster que potencialize as significativas vantagens de custo-desempenho dos microprocessadores produzidos em massa.

- Os multiprocessadores e clusters são altamente eficazes para workloads multiprogramados, que normalmente são o uso dominante dos mainframes e grandes servidores, bem como para servidores de arquivos ou servidores Web, que são um tipo restrito de workload paralelo. Quando um workload deseja compartilhar recursos, como um armazenamento de arquivos, ou pode compartilhar eficientemente o tempo de um recurso, como uma memória grande, um multiprocessador pode ser um host muito eficiente. Além disso, o sistema operacional necessário para executar workloads multiprogramados é comum.

- O uso de processamento paralelo em domínios como a computação científica e de engenharia é comum. Esse domínio de aplicação possui uma necessidade quase ilimitada de mais computação. Ele também possui muitas aplicações com uma grande quantidade de paralelismo natural. Entretanto, ele não tem sido fácil: Programar processadores paralelos até para essas aplicações continua sendo um desafio. Outra área de aplicação importante, que possui um mercado muito maior, são os sistemas de bancos de dados e processamento de transação de grande escala. Esse domínio de aplicação também tem um grande paralelismo natural disponível por meio de processamento paralelo de requisições independentes, mas suas necessidades para computação de grande escala, ao contrário do mero acesso a sistemas de armazenamento em grande escala, são menos compreendidas.

- O multiprocessamento on-chip parece estar crescendo em importância por duas razões: Primeiro, no mercado embutido em que o paralelismo natural normalmente existe, esses métodos são uma alternativa óbvia para processadores mais rápidos e possivelmente menos eficientes. Segundo, reduzir as devoluções no projeto de microprocessadores de topo de linha irá encorajar os projetistas a perseguirem o multiprocessamento on-chip como uma direção potencialmente mais lucrativa.

O futuro das arquiteturas MPP

Hennessy e Patterson deveriam passar os MPPs para o Capítulo 11.

Jim Gray, quando perguntado sobre a abordagem dos MPPs na segunda edição deste livro, aludindo à proteção de falência do Capítulo 11 da lei americana (1995)

Os multiprocessadores de pequena escala construídos usando esquemas de barramento de snooping são extremamente lucrativos. Tradicionalmente, os microprocessadores têm incluído muito da lógica para a coerência de cache no chip de processador, e vários permitem que os barramentos de dois ou mais processadores sejam conectados diretamente – implementando um barramento coerente sem qualquer lógica adicional. Com os níveis de integração modernos, os processadores múltiplos podem ser colocados dentro de um único die de silício, resultando em um multiprocessador altamente lucrativo. Os recentes microprocessadores têm incluído suporte para métodos NUMA, possibilitando conectarem quantidades de processadores pequenas a moderadas com pouco overhead.

O que não está claro hoje é como os processadores paralelos muito maiores serão construídos. As dificuldades enfrentadas pelos projetistas incluem o mercado relativamente pequeno para multiprocessadores muito grandes e a necessidade de que multiprocessadores que escalem para números maiores de processadores sejam extremamente lucrativos com números menores de processadores, no qual a maioria dos multiprocessadores será vendida. Parece haver quatro alternativas ligeiramente diferentes para multiprocessadores de grande escala:

1. Projetar um cluster usando *todos* os componentes encontrados facilmente no mercado, o que oferece o custo mais baixo. A força nesse método reside no uso de tecnologia de produto em todo lugar: nos processadores (PC ou nós de estação de trabalho), na interconexão (tecnologia de rede local de alta velocidade, como Gigabit Ethernet) e no software (sistemas operacionais padrão e linguagens de programação). É claro que esses multiprocessadores usarão troca de mensagens, e a comunicação provavelmente terá latência mais alta e largura de banda mais baixa do que os projetos alternativos. Para aplicações que não precisam de alta largura de banda ou comunicação de baixa latência, esse método pode ser bastante lucrativo. Os servidores Web, como o cluster do Google, são um bom exemplo.

2. Projetar computadores em cluster que usam nós de processador encontrados facilmente no mercado e uma interconexão personalizada. A vantagem desse projeto é a lucratividade do nó de processador padrão, que é um computador desktop reembalado; a desvantagem é que o modelo de programação provavelmente precisará ser o de troca de mensagens mesmo em um número de nós muito pequeno. O custo da interconexão personalizada pode ser significante e, portanto, encarecer o multiprocessador, em especial para pequenos números de nós. Um exemplo é o IBM SP.

3. Multiprocessadores de grande escala construídos por meio de clusters de multiprocessadores médios com combinações de tecnologias proprietárias e padrão para interconectar esses multiprocessadores. Esse método de cluster obtém sua lucratividade usando blocos de construção otimizados para o custo. Muitas empresas oferecem uma versão de topo de linha dessa máquina, incluindo a HP, a IBM e a Sun. Devido à natureza de dois níveis do projeto, o modelo de programação algumas vezes precisa ser mudado de memória compartilhada para troca de mensagens ou para uma variação diferente de memória compartilhada, entre os clusters. Essa classe de máquinas tem feito importantes avanços, especialmente nas aplicações comerciais.

4. Multiprocessadores de grande escala que crescem naturalmente, usando interconexão proprietária e tecnologia de controlador de comunicações. Existem duas dificuldades principais com esses projetos. Primeiro, os multiprocessadores não são lucrativos em pequenas escalas, nas quais o custo da escalabilidade não é avaliado. Segundo, esses multiprocessadores possuem modelos de programação que são incompatíveis, em variados níveis, com os multiprocessadores menores e médios convencionais. O SGI Origin é um exemplo.

Cada um desses métodos possui vantagens e desvantagens e a importância das limitações de qualquer método é dependente do tipo de aplicação. Não se sabe ao certo qual vencerá como multiprocessador de grande escala, embora o crescimento do mercado para servidores Web tenha tornado os "racks de PCs" a forma dominante, pelo menos pelo número de sistemas.

O futuro das arquiteturas de multiprocessadores

Como vimos no Capítulo 6, os arquitetos estão usando técnicas ainda mais complexas para tentar explorar mais paralelismo em nível de instrução. Os candidatos para encontrar quantidades cada vez maiores de paralelismo em nível de instrução de uma maneira eficiente de explorar são um tanto limitados. Como abordado no Capítulo 7, existem problemas crescentes cada vez mais difíceis de serem resolvidos na construção de hierarquias de memória para processadores de alto desempenho. Naturalmente, as constantes melhorias de tecnologia nos permitirão continuar a avançar na velocidade de clock. Entretanto, o uso das melhorias de tecnologia que permitem uma maior velocidade de porta, por si só, não é suficiente para manter o incrível crescimento de desempenho que a indústria tem experimentado por mais de 20 anos. Além disso, enquanto a potência aumenta mais de 100 watts por chip, não se sabe o quanto mais alto ela pode ir nos sistemas refrigerados a ar. Portanto, a potência pode acabar sendo outro limite ao desempenho.

Infelizmente, por mais de uma década, os aumentos no desempenho vieram à custa de limitações cada vez maiores no uso da área de silício, conexões externas e potência. Esse fenômeno de redução de retornos apenas recentemente pareceu ter diminuído a velocidade do crescimento de desempenho. O que está claro é que não podemos sustentar a alta velocidade na melhoria de desempenho sem inovações significativas na arquitetura de computadores.

Com isso em mente, em 2004 parecia que a direção de longo prazo seria usar silício aumentado para produzir múltiplos processadores em um único chip. Essa direção é atraente do ponto de vista da arquitetura – ela oferece uma maneira de escalar o desempenho sem aumentar a complexidade do hardware. Ela também oferece um meio de facilitar algumas das dificuldades no projeto de sistema de memória, já que uma memória distribuída pode ser usada para escalar largura de banda enquanto mantém baixa latência para acessos locais. Finalmente, os processadores redundantes podem ajudar com a confiança. O problema está no software e em que inovações de arquitetura podem ser usadas para tornar o software mais fácil.

Em 2000, a IBM anunciou os primeiros chips comerciais com dois processadores de uso geral em um único die de silício, o processador Power4. Cada Power4 contém dois processadores, uma cache secundária compartilhada, uma interface para uma cache terciária off-chip ou para a memória principal, e um sistema de comunicação chip a chip, que permite que um módulo de quatro processadores conectados por crossbar seja construído sem lógica adicional. Usando quatro chips Power4 e as DRAMs apropriadas, um sistema de oito processadores pode ser integrado em uma placa de aproximadamente 20,32cm de lado. Em 2002, a Sun anunciou um chip com quatro núcleos de processador em um chip, cada um multithreaded suportando oito threads, dando ao programador a sensação de 32 processadores. Os sistemas que usam esse chip devem ser comercializados em 2005. Em 2004, a Intel anunciou chips de dois processadores.

Se o número de processadores por chip crescer conforme a Lei de Moore, dezenas de processadores serão possíveis no futuro próximo. O desafio para esses "micromultiprocessadores" é a base de software que poderá explorá-los, o que pode levar a oportunidades de inovação na representação e otimização dos programas.

Mudanças de paradigma na indústria de computadores

A Figura 9.10.1 mostra o que queremos dizer com o *espectro evolução-revolução* da inovação da arquitetura de computadores. À esquerda estão idéias invisíveis ao usuário (presumivelmente, excetuando o melhor custo, o melhor desempenho ou ambos) e estão no lado evolucionário do espectro. No

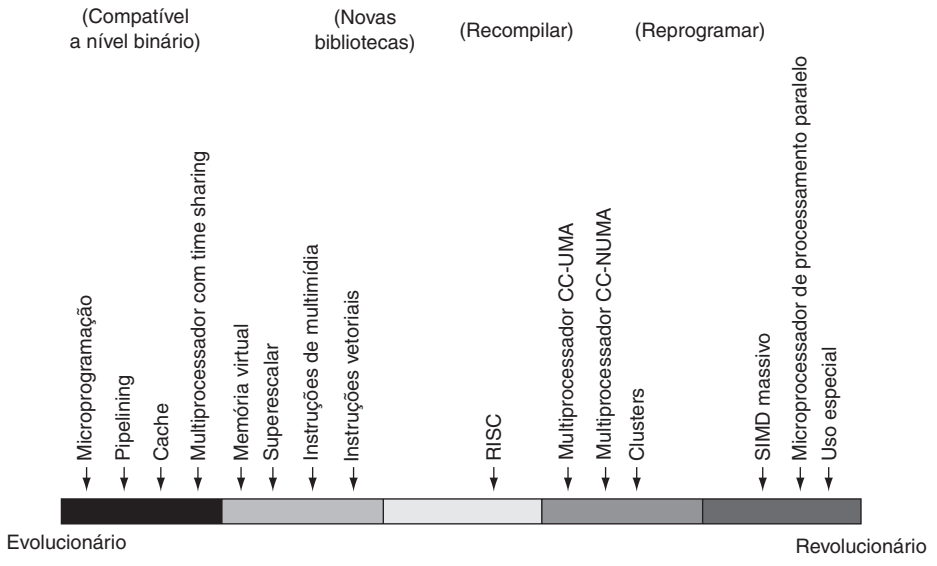


FIGURA 9.10.1 O espectro evolução-revolução da arquitetura de computador. As quatro primeiras colunas diferem da última coluna na medida em que as aplicações e os sistemas operacionais podem ser portados de outros computadores em vez de serem escritos desde o início. Por exemplo, o RISC está listado no centro do espectro porque a compatibilidade do usuário está apenas no nível das linguagens de alto nível (HLLs), enquanto a microprogramação permite compatibilidade binária e os multiprocessadores de processamento paralelo exigem mudanças nos algoritmos e HLLs extensíveis. Você pode ver vários tipos de multiprocessadores nessa figura. "Multiprocessador com time sharing" significa multiprocessadores justificados por executar muitos programas independentes ao mesmo tempo. "CC-UMA" e "CC-NUMA" significam multiprocessadores UMA e NUMA com coerência de cache executando subsistemas paralelos como bancos de dados ou servidores de arquivos. Além disso, as mesmas aplicações são destinadas à "troca de mensagens". "Multiprocessador de processamento paralelo" significa um multiprocessador de algum tipo vendido para acelerar programas individuais desenvolvidos por usuários. (Veja a Seção 9.11 para aprender mais sobre SIMD.)

outro lado estão idéias de arquitetura revolucionárias. Essas são as idéias que exigem novas aplicações por parte dos programadores, que precisam aprender novas linguagens de programação e modelos de computação e precisam inventar novas estruturas de dados e algoritmos.

As idéias revolucionárias são mais fáceis de nos empolgar do que as idéias evolucionárias, mas, para serem adotadas, precisam ter uma compensação muito mais alta. As caches são um exemplo de melhoria evolucionária. Dentro de cinco anos após a primeira publicação sobre caches, quase todas as empresas de computador estavam projetando um computador com uma cache. As idéias de RISC estavam mais próximas do centro do espectro, pois levou mais de oito anos para que a maioria das empresas tivesse um produto RISC. A maioria dos multiprocessadores tendeu para o lado revolucionário do espectro, com os multiprocessadores de larga escala (MPPs) sendo mais revolucionários do que outros.

O desafio para os projetistas de hardware e software que propõem que os multiprocessadores e o processamento paralelo se tornassem a regra, e não a exceção, é o rompimento da base estabelecida dos programas. Existem duas maneiras possíveis para que essa mudança de paradigma fosse facilitada: se o processamento paralelo oferecer a única alternativa para melhorar o desempenho, e se os avanços na tecnologia de hardware e software puderem construir uma rampa suave que permita que o movimento para o processamento paralelo, pelo menos com pequenos números de processadores, seja mais evolucionário. Talvez o custo/desempenho seja substituído por novas metas de estabilidade, segurança e/ou custo reduzido de propriedade como a principal justificativa de uma mudança desse tipo.

Quando contemplar o futuro – e quando inventar suas próprias contribuições para o campo –, lembre-se da interface hardware/software. A aceitação de idéias de hardware exige aceitação pelas pessoas de software; portanto, as pessoas de hardware precisam aprender mais sobre software. Além

disso, se as pessoas de software quiserem boas máquinas, elas precisam aprender mais sobre hardware para serem capazes de se comunicar com – e, portanto, influenciar – os projetistas de hardware. Também tenha em mente os princípios da organização de computadores encontrados neste livro; eles certamente guiarão os computadores do futuro, exatamente como guiaram os computadores do passado.

9.11 Perspectiva histórica e leitura adicional

Como o paralelismo pode aparecer em muitos níveis, é útil categorizar as alternativas. Em 1966, Flynn propôs um modelo simples de categorizar computadores que é amplamente usado. Examinando o componente mais restrito da máquina, ele contou o número de fluxos de instruções paralelas e de dados e, depois, rotulou o computador com este resultado:

1. *Fluxo de instruções único, fluxo de dados único* (SISD, o uniprocessador)
2. *Fluxo de instruções único, fluxos de dados múltiplos* (SIMD)
3. *Fluxos de instruções múltiplos, fluxo de dados único* (MISD)
4. *Fluxos de instruções múltiplos, fluxos de dados múltiplos* (MIMD)

Naturalmente, algumas máquinas são híbridas dessas categorias, mas esse modelo clássico sobreviveu porque é simples, fácil de entender e oferece uma boa estimativa inicial. Ele também é – talvez por ser inteligível – o esquema mais utilizado.

Sua primeira pergunta sobre o modelo deve ser: "Único ou múltiplo comparado com o quê?" Uma máquina que soma um número de 32 bits em 1 ciclo de clock pareceria ter fluxos de dados múltiplos quando comparada com um computador de bits seriais que leva 32 ciclos de clock para somar. Flynn escolheu computadores populares para a época, o IBM 704 e o IBM 7090, como o modelo do SISD; hoje, as implementações MIPS nos Capítulos 5 e 6 seriam bons pontos de referência.

Computadores SIMD

Os computadores SIMD operam em vetores de dados. Por exemplo, quando uma única instrução SIMD soma 64 números, o hardware SIMD envia 64 fluxos de dados para 64 ALUs para formar 64 somas dentro de um único ciclo de clock.

O mérito do SIMD é que todas as unidades de execução paralelas são sincronizadas e respondem a uma única instrução que emana de um único contador de programa (PC). Da perspectiva de um programador, isso está próximo do já familiar SISD. Embora cada unidade esteja executando a mesma instrução, cada unidade de execução tem seus próprios registradores de endereço e, portanto, cada unidade pode ter diferentes endereços de dados.

A motivação original por trás do SIMD foi amortizar o custo da unidade de controle entre dezenas de unidades de execução. Outra vantagem é o tamanho reduzido da memória de programa – o SIMD precisa apenas de uma cópia do código executado simultaneamente, enquanto o MIMD pode precisar de uma cópia em cada processador. A memória virtual e a crescente capacidade dos chips DRAM reduziram a importância dessa vantagem.

Os computadores SIMD reais possuem uma mistura de instruções SISD e SIMD. Normalmente há um computador host SISD para realizar operações seqüenciais como desvios ou cálculos de endereços. As instruções SIMD são difundidas para todas as unidades de execução, cada qual com seu próprio conjunto de registradores e memória. As unidades de execução se baseiam nas redes de interconexão para a troca de dados.

paralelismo de dados Paralelismo obtido tendo dados massivos.

O SIMD funciona melhor quando lida com arrays em loops *for*. Portanto, para que o paralelismo massivo funcione no SIMD, precisa haver dados massivos, ou **paralelismo de dados**. O SIMD tem

Instituição	Nome	Número máximo de processadores	Bits/ proc.	Velocidade de clock do proc. (MHz)	Número de FPUs	Tamanho/ sistema de memória máximo (MB)	BW/sistema de comunicações (MB/s)	Ano
Universidade de Illinois	Illiac IV	64	64	13	64	1	2.560	1972
ICL	DAP	4.096	1	5	0	2	2.560	1980
Goodyear	MPP	16.384	1	10	0	2	20.480	1987
Thinking Machines	CM-2	65.536	1	7	2048 (opcional)	512	16.384	1987
Maspar	MP-1216	16.384	4	25	0	256 ou 1.024	23.000	1989

FIGURA 9.11.1 Características de cinco computadores SIMD. Número de FPUs significa o número de unidades de ponto flutuante.

seu lado mais fraco em instruções *case* ou *switch*, onde cada unidade de execução precisa realizar uma operação diferente em seus dados, dependendo de que dados ela possui. As unidades de execução com os dados errados são desativadas para que as unidades com dados corretos possam continuar. Essas situações são executadas em 1/*n* do desempenho, onde *n* é o número de cases.

Um compromisso básico nas máquinas SIMD é o desempenho do processador *versus* o número de processadores. O Connection Machine 2 (CM-2), por exemplo, oferece 65.536 processadores com largura de um único bit, enquanto o Illiac IV tinha 64 processadores de 64 bits. A Figura 9.11.1 lista as características de alguns computadores SIMD conhecidos.

Seguramente, o Illiac IV (visto na Figura 9.11.2) é o mais famoso dos projetos de supercomputador. Embora bem-sucedido em promover várias tecnologias úteis em projetos posteriores, o Illiac IV falhou

FIGURA 9.11.2 A unidade de controle do Illiac IV seguida de seus 64 elementos de processamento. Esse talvez tenha sido o mais famoso dos supercomputadores. O projeto iniciou em 1965 e executou sua primeira aplicação real em 1976. Os 64 processadores usavam um clock de 13MHz e o tamanho de sua memória principal combinada era de 1MB: 64 16KB. O Illiac IV foi a primeira máquina a nos ensinar que o software para máquinas paralelas domina os problemas de hardware. A foto é cortesia do Ames Research Center da NASA.

como um computador. Os custos aumentaram de US$8 milhões estimados em 1966 para US$31 milhões em 1972, apesar da construção de apenas um quarto da máquina planejada. O desempenho real foi no máximo de 15 MFLOPS comparados com a previsão inicial de 1.000 MFLOPS para o sistema completo (veja Falk [1976]). Entregue para a Ames Research da NASA em 1972, o computador levou mais três anos de engenharia para que se tornasse operacional. Felizmente, os arquitetos de computadores não desanimam facilmente; os sucessores SIMD do Illiac IV incluem o ICL DAP, Goodyear MPP (Figura 9.11.3), Thinking Machines CM-1 e CM-2 e Maspar MP-1 e MP-2.

Computadores vetoriais

processamento vetorial Um modelo de arquitetura e compilador popularizado pelos supercomputadores e em que operações de alto nível atuam em arrays de números lineares.

Um modelo relacionado ao SIMD é o **processamento vetorial**. Trata-se de um modelo bem-estabelecido de arquitetura e compilador popularizado pelos supercomputadores e que é consideravelmente mais usado do que o SIMD. Os processadores vetoriais possuem operações de alto nível que atuam em arrays de números lineares, ou vetores. Um exemplo de operação vetorial é

$$A = B \times C$$

onde A, B e C são vetores de 64 elementos de números de ponto flutuante de 64 bits. O SIMD possui instruções semelhantes; a diferença é que os processadores vetoriais dependem de unidades funcionais em pipeline que normalmente operam em alguns elementos do vetor por ciclo de clock, enquanto o SIMD geralmente opera em todos os elementos ao mesmo tempo.

As vantagens dos computadores vetoriais sobre os processadores SISD tradicionais incluem as seguintes:

- Cada resultado é independente dos resultados anteriores, o que permite pipelines profundos e altas velocidades de clock.
- Uma única instrução vetorial realiza uma grande quantidade de trabalho, o que significa menos buscas de instruções em geral e menos instruções de desvio e, portanto, menos desvios mal previstos.
- As instruções vetoriais acessam a memória um bloco por vez, o que permite que a latência de memória seja amortizada entre, digamos, 64 elementos.

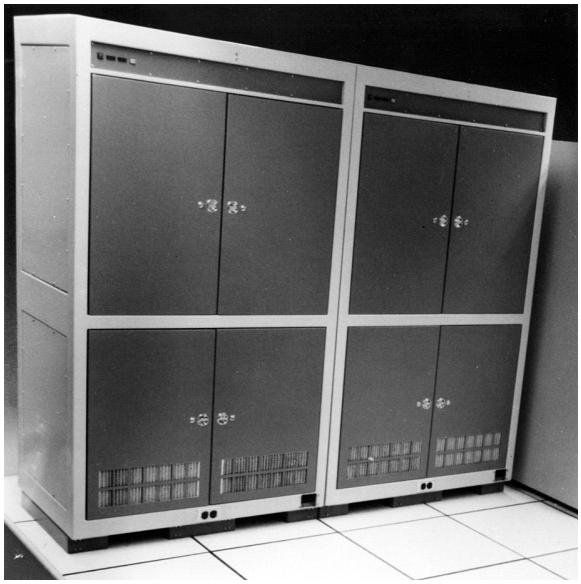


FIGURA 9.11.3 A O Goodyear MPP com 16.384 processadores. Ele foi entregue em 2 de maio de 1983 para o NASA Goddard Space Center e começou a operar no dia seguinte. Ele foi aposentado em 1º de março de 1991.

- As instruções vetoriais acessam a memória com padrões conhecidos, o que permite que vários bancos de memória forneçam operandos simultaneamente.

Essas duas últimas vantagens significam que os processadores vetoriais não precisam se basear em altas taxas de acerto das caches de dados para ter alto desempenho. Eles costumam se basear em memórias de baixa latência, normalmente compostas por SRAM, e possuem 1.024 bancos de memória para alcançar alta largura de banda de memória.

Surpreendentemente, os vetores também encontraram uso nas aplicações de mídia.

SIMD em multimídia

Infelizmente, os grupos de marketing tomaram o acrônimo SIMD emprestado para descrever uma idéia simples: Se os dados de sua aplicação são curtos, em vez de realizar uma operação por ciclo de clock, use registradores e ALUs largos para realizar muitas operações curtas por ciclo de clock. Por exemplo, com dados de 8 bits e registradores e ALUs de 64 bits, um processador pode realizar 8 operações por ciclo de clock. Se a ALU pode suprimir os "carries" entre grupos de 8 bits, ela pode realizar somas e subtrações curtas em paralelo. Embora esse uso do SIMD seja mal denominado, ele é bastante comum.

Esse projeto, na verdade, é um subconjunto de uma arquitetura vetorial restrita, carente de alguns dos importantes recursos e da elegância do projeto vetorial completo: tornar o número máximo de elementos por vetor independente da arquitetura, de loads e de stores de índices etc.

Detalhamento: embora o MISD faça parte da classificação de Flynn, ele é difícil de imaginar. Um único fluxo de instrução é mais simples do que múltiplos fluxos de instrução, mas múltiplos fluxos de instrução com múltiplos fluxos de dados (MIMD) são mais fáceis de imaginar do que múltiplas instruções com um único fluxo de dados (MISD).

Computadores MIMD

É difícil distinguir o primeiro MIMD: Os argumentos para as vantagens da execução paralela podem ser seguidos até o século XIX [Menabrea, 1842]! Além disso, até mesmo o primeiro computador da Eckert-Mauchly Corporation tinha unidades duplicadas, nesse caso para melhorar a confiabilidade.

Dois dos projetos de multiprocessador mais bem-documentados foram executados na década de 1970 na Carnegie-Mellon University. O primeiro foi o C.mmp, constituído por 16 PDPs-11 conectados por uma chave crossbar com 16 unidades de memória. Ele estava entre os primeiros multiprocessadores com mais de alguns processadores e tinha um modelo de programação de memória compartilhada. Muito do foco da pesquisa no projeto C.mmp estava no software, especialmente na área de sistemas operacionais. Uma máquina posterior, o Cm*, era um multiprocessador baseado em clusters com uma memória distribuída e um tempo de acesso não uniforme, que tornava a programação ainda mais problemática. A ausência de caches e a longa latência de acesso remoto tornavam o posicionamento dos dados um fator vital.

Embora mainframes muito grandes tenham sido construídos com processadores múltiplos nos anos 70, os multiprocessadores não se tornaram altamente bem-sucedidos até a década de 1980. Bell [1985] sugere que o segredo para o sucesso foi que o menor tamanho do microprocessador permitiu que o barramento de memória substituísse o hardware da rede de interconexão, e que os sistemas operacionais portáveis significavam que os projetos de processadores paralelos não mais exigiam a invenção de um novo sistema operacional. Ele distingue processadores paralelos com endereços privados múltiplos chamando-os de **multicomputadores**, reservando o termo **multiprocessador** para máquinas com um único espaço de endereçamento. Portanto, Bell classificaria um cluster como um multicomputador.

multicomputador Processadores paralelos com múltiplos endereços privados.

multiprocessador Processadores paralelos com um único endereçamento compartilhado.

O primeiro multiprocessador conectado a barramento com caches com snooping foi o Synapse N+1 em 1984. Os meados da década de 1980 viram uma explosão no desenvolvimento de protocolos

de coerência alternativos, e Archibald e Baer [1986] forneceram uma boa pesquisa e análise, bem como referências aos documentos originais. O final da década de 1980 trouxe a introdução de muitas arquiteturas de cache com snooping conectadas a barramento, incluindo o Silicon Graphics 4D/240, o Encore Multimax e o Sequent Symmetry.

No esforço de construir multiprocessadores de grande escala, duas direções diferentes foram tomadas: os multicomputadores de troca de mensagens e os multiprocessadores de memória compartilhada escalável. Embora tenha havido muitas tentativas de construir multiprocessadores conectados a hipercubos e a malhas, uma das primeiras máquinas a montar com sucesso todas as peças foi o Cosmic Cube, construído na Caltech [Seitz, 1985]. Ele introduziu importantes avanços na tecnologia de roteamento e interconexão e reduziu substancialmente o custo da interconexão, o que ajudou a viabilizar o multicomputador. As máquinas comerciais com arquiteturas relacionadas incluíam o Intel iPSC 860, o Intel Paragon e o Thinking Machines CM-5. Infelizmente, o mercado para essas máquinas provou ser muito menor do que o esperado, e a Intel saiu do negócio (com o ASCI Red sendo sua última máquina) e a Thinking Machines não mais existe. Hoje, esse espaço é principalmente de clusters, como o IBM RS/6000 SP2 (Figura 9.11.4).

Estender o modelo de memória compartilhada com coerência de cache escalável foi feito combinando diversas idéias. As técnicas baseadas em diretórios para coerência de cache foram realmente conhecidas antes das técnicas de cache com snooping. Na verdade, o primeiro protocolo de coerência de cache usava diretórios e era usado no IBM 3081 em 1976. A idéia de distribuir diretórios com memórias para obter uma implementação escalável da coerência de cache (agora chamada de memória compartilhada distribuída, ou DSM) foi a base para o multiprocessador Stanford DASH; ele é considerado o precursor dos computadores NUMA.

Existe uma grande quantidade de informações sobre multiprocessadores: conferências, documentos periódicos e mesmo livros aparecem regularmente. Uma boa fonte é a Supercomputing Conference, realizada anualmente desde 1988. Dois grandes periódicos, o *Journal of Parallel and Distributed Computing* e o *IEEE Transactions on Parallel and Distributed Systems*, contêm extensos documentos sobre aspectos da computação paralela. Livros sobre computação paralela foram escritos por Almasi e Gottlieb [1989]; Andrews [1991]; Culler, Singh e Gupta [1998]; e Hwang [1993]. O livro de Pfister [1998] é um dos poucos sobre clusters.

FIGURA 9.11.4 O IBM RS/6000 SP2 com 256 processadores. Esta máquina de memória distribuída é construída usando placas de computadores desktop inalteradas, além de um switch personalizado como a interconexão. Comparado com o SP2, a maioria dos clusters usa uma rede local comutada original. A foto é cortesia do Lawrence Livermore National Laboratory.

Detalhamento: embora fosse concebível escrever 100 programas diferentes para 100 diferentes processadores em uma máquina MIMD, na prática, isso mostrou ser impossível. Hoje, os programadores de MIMD escrevem um único programa de origem e pensam no mesmo programa sendo executado em todos os processadores. Esse método às vezes é chamado de *programa único e dados múltiplos* (SPMD).

Leitura complementar

Almasi, G. S. e A. Gottlieb [1989]. *Highly Parallel Computing*, Benjamin/Cummings, Redwood City, CA.

Um livro-texto abordando computadores paralelos.

Amdahl, G. M. [1967]. "Validity of the single processor approach to achieving large scale computing capabilities", *Proc. AFIPS Spring Joint Computer Conf.*, Atlantic City, NJ (Abril), 483–85.

Escrito em resposta às reclamações do Illiac IV, este artigo de três páginas descreve a Lei de Amdahl e fornece a resposta clássica aos argumentos para abandonar a forma de computação atual.

Andrews, G. R. [1991]. *Concurrent Programming: Principles and Practice,* Benjamin/Cummings, Redwood City, CA.

Um texto que fornece os princípios da programação paralela.

Archibald, J. e J.-L. Baer [1986]. "Cache coherence protocols: Evaluation using a multiprocessor simulation model", *ACM Trans. on Computer Systems* 4:4 (Novembro), 273–98.

Documento de pesquisa clássico dos protocolos de coerência de cache com barramento compartilhado.

Arpaci-Dusseau, A., R. Arpaci-Dusseau, D. Culler, J. Hellerstein e D. Patterson [1997]. "Highperformance sorting on networks of workstations", *Proc. ACM SIGMOD/PODS Conference on Management of Data*, Tucson, AZ Maio, 12–15.

Como uma ordenação de registros mundiais foi realizada em um cluster, incluindo crítica da arquitetura da estação de trabalho e da interface de rede. Em 1º de abril de 1997, eles aumentaram o recorde para 8,6GB em 1 minuto e 2,2 segundos para ordenar 100MB.

Bell, C. G. [1985]. "Multis: A new class of multiprocessor computers", *Science* 228 (26 de abril), 462–67.

Distingue multiprocessadores de endereçamento compartilhado e de endereçamento não compartilhado com base nos microprocessadores.

Culler, D. E. e J. P. Singh, com A. Gupta [1998]. *Parallel Computer Architecture*, Morgan Kaufmann, San Francisco.

Um livro-texto sobre computadores paralelos.

Falk, H. [1976]. "Reaching for the Gigaflop", *IEEE Spectrum* 13:10 (Outubro), 65–70.

Critica a triste história do Illiac IV: quatro vezes o custo e menos de um décimo do desempenho das metas originais.

Flynn, M. J. [1966]. "Very high-speed computing systems", *Proc. IEEE* 54:12 (Dezembro), 1901–09.

Artigo clássico mostrando classificações SISD/SIMD/MISD/MIMD.

Hennessy, J. e D. Patterson [2003]. Capítulos 6 e 8 em *Arquitetura de Computadores: Uma abordagem quantitativa,* terceira edição, Editora Campus, Rio de Janeiro.

Uma cobertura mais aprofundada sobre uma variedade de tópicos de multiprocessador e cluster, incluindo programas e medições.

Hord, R. M. [1982]. *The Illiac-IV, the First Supercomputer,* Computer Science Press, Rockville, MD.

Uma contabilidade histórica do projeto Illiac IV.

Hwang, K. [1993]. *Advanced Computer Architecture with Parallel Programming*, McGraw-Hill, New York.

Outro livro-texto abordando os computadores paralelos.

Kozyrakis, C. e D. Patterson [2003]. "Scalable vector processors for embedded systems", *IEEE Micro* 23:6 (novembro a dezembro), 36–45.

Exame de uma arquitetura vetorial para o conjunto de instruções MIPS em processamento de mídia e sinais

Menabrea, L. F. [1842]. "Sketch of the analytical engine invented by Charles Babbage", *Bibliothèque Universelle de Genève* (Outubro).

Com certeza, a primeira referência sobre multiprocessadores, este matemático fez este comentário enquanto traduzia artigos sobre o computador mecânico de Babbage.

Pfister, G. F. [1998]. *In Search of Clusters: The Coming Battle in Lowly Parallel Computing*, segunda edição, Prentice-Hall, Upper Saddle River, NJ.

Um interessante livro que defende os clusters e critica os multiprocessadores NUMA.

Seitz, C. [1985]. "The Cosmic Cube", *Comm. ACM* 28:1 (Janeiro), 22–31.

Um artigo tutorial sobre um processador paralelo conectado por meio de uma hiper-árvore. O Cosmic Cube é o ancestral dos supercomputadores Intel.

Slotnick, D. L. [1982]. "The conception and development of parallel processors–A personal memoir", *Annals of the History of Computing* 4:1 (Janeiro), 20–30.

Memórias do início do processamento paralelo pelo arquiteto do Illiac IV.

9.12 Exercícios

9.1 [15] <§9.1> Escreva um artigo de uma página examinando em sua vida situações em que a cooperação está presente e a exclusão mútua é obtida. Você pode querer considerar coisas como estradas passando de duas pistas para uma, esperar em filas em diferentes tipos de empresas, obter a atenção do seu professor para fazer perguntas etc. Tente descobrir diferentes meios e mecanismos usados para comunicação e sincronização. Existe alguma situação em que você desejaria que um algoritmo diferente fosse usado para que a latência ou a largura de banda fosse melhorada, ou talvez o sistema fosse mais "justo"?

9.2 [10] <§9.1> Considere as seguintes partes de dois programas diferentes executados ao mesmo tempo em dois processadores em um multiprocessador simétrico (SMP). Considere que, antes deste código ser executado, `x` e `y` são 0.

Processador 1: `...; x = x + 2; y = x + y; ...`

Processador 2: `...; y = x + 2; ...`

Quais são os possíveis valores resultantes de `x` e `y`, considerando que o código é implementado usando uma arquitetura load-store? Para cada resultado possível, explique como `x` e `y` poderiam obter esses valores. (Dica: você precisa examinar todas as intercalações possíveis das instruções assembly.)

9.3 [10] <§§9.1–9.3> Imagine que todos os funcionários de uma enorme empresa tenham esquecido quem dirige a empresa e só podem lembrar para quem eles trabalham. A gerência está considerando se deve emitir uma das seguintes instruções:

- "Hoje, todos os funcionários devem perguntar ao seu chefe quem ele é; amanhã, pergunte a essa pessoa quem é o chefe *dela* e assim por diante, até descobrir quem dirige a empresa."
- "Por favor, todos escrevam o nome do seu chefe em uma folha de papel. Descubra qual é o nome da pessoa na folha de papel *dele* e, amanhã, escreva o nome em sua folha de papel antes de vir para o trabalho. Repita esse processo até descobrir quem dirige e empresa."

Escreva um parágrafo descrevendo a diferença entre essas duas instruções e os resultados. Explique a relação entre as duas alternativas descritas anteriormente e o que você aprendeu sobre sincronização e comunicação cooperativas entre processos.

9.4 [5] <§§9.1–9.3> {Ex. 9.3} Analise o desempenho dos dois algoritmos anteriores. Considere empresas contendo 64 pessoas, 1.024 pessoas ou 16.384 pessoas. Você pode imaginar alguma maneira de melhorar um dos dois algoritmos ou de realizar a mesma tarefa ainda mais rápido?

9.5 [5] <§9.3> Conte o número de transações no barramento para a seguinte seqüência de atividades envolvendo dados compartilhados. Considere que os dois processadores usam cache write-back, coerência de cache com atualização de escrita e um tamanho de bloco de uma word. Considere também que todas as words nos dois casos são limpas.

Etapa	Processador	Atividade de memória	Endereço de memória
1	Processador 1	Escrita	100
2	Processador 2	Escrita	104
3	Processador 1	Leitura	100
4	Processador 2	Leitura	104
5	Processador 1	Leitura	104
6	Processador 2	Leitura	100

9.6 [10] <§9.3> O falso compartilhamento pode levar a um tráfego de barramento e atrasos desnecessários. Siga as instruções para o Exercício 9.5, mas, desta vez, mude o tamanho de bloco para quatro words.

9.7 [15] <§9.6> Outra topologia de rede possível é uma grade tridimensional. Desenhe a topologia na Figura 9.6.1 para 64 nós. Qual é a largura de banda de bisseção dessa topologia?

9.8 [1 semana] <§§9.2–9.6> Um processador paralelo normalmente é comercializado usando programas que podem escalar o desempenho linearmente com o número de processadores. Porte os programas escritos para um processador paralelo para outro e meça seu desempenho absoluto e como ele muda conforme você muda o número de processadores. Que alterações precisam ser feitas para melhorar o desempenho dos programas portados em cada máquina? Qual é o desempenho de acordo com cada programa?

9.9 [1 semana] <§§9.2–9.6> Em vez de tentar criar benchmarks justos, invente programas que façam um processador paralelo parecer excelente comparado com os outros e também programas que sempre façam um parecer melhor do que os outros. Quais são as principais características de desempenho de cada programa e máquina?

9.10 [1 semana] <§§9.2–9.6> Os processadores paralelos normalmente mostram aumentos de desempenho enquanto você aumenta o número de processadores, com o ideal sendo n vezes o speedup para n processadores. O objetivo deste exercício é criar um benchmark tendencioso que obtém o pior desempenho quando você acrescenta processadores. Por exemplo, um processador no processador paralelo executaria o programa mais rápido, dois seriam mais lentos, quatro seriam mais lentos do que dois e assim por diante. Quais são as principais características de desempenho para cada organização que fornece speedup linear inverso?

9.11 [1 semana] <§§9.2–9.6> As estações de trabalho em rede podem ser consideradas processadores paralelos, embora com comunicação lenta relativa à computação. Porte benchmarks de processadores paralelos para uma rede usando chamadas de procedimento remoto para comunicação. Até que ponto os benchmarks escalam bem na rede comparado com o processador paralelo? Quais são as diferenças práticas entre estações de trabalho em rede e um processador paralelo comercial?

9.12 [1 semana] <§§9.2–9.6> Melhoria de desempenho *superlinear* significa que um programa em n processadores é mais de n vezes mais rápido do que o uniprocessador equivalente. O argumento para o speedup superlinear é que o tempo gasto servindo interrupções ou trocando contextos é reduzido quando você tem muitos processadores, pois apenas um deles precisa servir interrupções e há mais processadores para serem compartilhados pelos usuários. Meça o tempo gasto em um workload

para manipular interrupções ou troca de contexto para um uniprocessador comparado com um processador paralelo. Esse workload pode ser uma combinação de tarefas independentes para um ambiente de multiprogramação ou uma única tarefa grande. O argumento se mantém válido?

9.13 [15] <§9.10> Construa um cenário por onde uma arquitetura verdadeiramente revolucionária – escolha sua candidata favorita – irá desempenhar um papel significativo. "Significativo" é definido como 10% dos computadores vendidos, 10% dos usuários, 10% do dinheiro gasto em computadores ou 10% de alguma outra cifra importante.

9.14 [20] <§§9.1–9.10> Este capítulo introduziu muitos termos de vocabulário novos relacionados ao tema dos multiprocessadores. Alguns dos exercícios deste capítulo (por exemplo, os Exercícios 9.1 e 9.3) são baseados em uma analogia em que pessoas são imaginadas como processadores e grupos de pessoas como multiprocessadores. Escreva um artigo de uma página explorando essa analogia em mais detalhes. Por exemplo, existem grupos de pessoas que usam técnicas semelhantes à troca de mensagens ou às memórias compartilhadas? Você poderia criar analogias para protocolos de coerência de cache, topologias de rede ou clusters? Procure incluir pelo menos um termo de vocabulário de cada seção do texto.