Bem vindo ao esboço das notas de aula do professor Juliano Mourão Vieira para a disciplina de Arquitetura e Organização de Computadores na UTFPR.

MICROPROCESSADORES, SEU PROJETO & TECNOLOGIAS

Sumário

O Que É Arquitetura de Computadores?

Podemos definir simplificadamente Arquitetura de Computadores como sendo o estudo das ISAs e das microarquiteturas, e incluir aí também os periféricos e ligações respectivas. Mas o foco central da disciplina são as escolhas que podemos fazer a respeito do microprocessador, e como implementá-las.

Uma ISA (Instruction Set Architecture) é nominalmente a arquitetura que escolhemos para o nosso conjunto de instruções. Isso inclui todas as instruções que nosso processador deverá executar, e seus efeitos, mas não apenas isso. O nome pode ser meio confuso, pois a ISA efetivamente é uma especificação completa de todos os comportamentos observáveis do processador. Uma ISA inclui descrições de:

- os registradores disponíveis, suas funções e número de bits;
- todas as instruções, seu funcionamento (inclusive limites) e sua codificação (seus opcodes);

- comportamento do processador no momento em que é energizado (reset a frio ou a quente);
- interrupções disponíveis e resposta a elas;
- exceções (erros) e comportamento nestes casos;
- tratamento com os periféricos;
- temporização relativa ao clock e execução de instruções, se aplicável;
- etc.

Portanto podemos dizer que a ISA é a especificação completa do comportamento do processador.

Mas se a ISA é a especificação, a microarquitetura é a implementação. Por exemplo, há diversas formas de se implementar um circuito somador: em disciplinas anteriores, você deve ter visto os somadores completos e meio-somadores cascateados, uma estratégia simples e facilmente expansível, mas bastante lenta no pior caso, com carrys sendo propagados do LSB para o MSB, custando atrasos de portas lógicas em cada uma das etapas. O mais rápido seria fazer uma tabela em memória, mas para somar dois números de 8 bits a tabela teria 64 kbytes para produzir o resultado, o que consumiria bastante espaço da pastilha de silício. Modernamente, utiliza-se soma com carry-lookahead, que é bastante rápida no caso médio e possui um circuito razoavelmente compacto (olhe lá no livro-texto se quiser ver o funcionamento).

Então, conforme o objetivo do processador em questão, podemos escolher estratégias mais rápidas que consomem mais energia ou estratégias mais lentas e econômicas. Veja o caso da figura:

`FIGURA: [Figura Phenom vs Atom]`

Note que ambos processadores, de 2008, possuem a mesma ISA, x86, e portanto podem rodar exatamente os mesmos programas, sem qualquer alteração: o executável binário que roda no Phenom roda também no Atom. A diferença é que o Phenom foi criado para ter alto desempenho, não importando muito o consumo, portanto é um quadcore que consome 125W, enquanto que o Atom foi criado para economia, provavelmente visando a aumentar duração da bateria de celulares e netbooks, portanto tem apenas um core consumindo míseros 2W. A diferença é só na microarquitetura.

Outro exemplo interessante é a arquitetura big.LITTLE proposta pela ARM para seus processadores. Nela, um core inclui dois processadores que nunca são usados simultaneamente: um é para momentos de pouca exigência de desempenho, que consome pouco. Caso o sistema exija muito desempenho por um certo período, este processador é desligado e a execução é passada para o outro processador, muito mais rápido mas que consome muito mais. Quando o pico de desempenho passa, o controle volta ao processador econômico.

`FIGURA: [Figura big.LITTLE]`

A ideia é possuir um balanço entre baixo consumo e alto desempenho, já que tipicamente temos "rajadas" de grande exigência do processador, intercaladas com longos períodos de ociosidade ou workload (volume de trabalho) pequeno. O chaveamento entre os processadores é praticamente invisível para o programa sendo executado, já que a ISA de ambos os processadores é a mesma.

Harvard vs von Neumann

A primeira decisão arquitetural descrita nos livros-texto diz respeito à organização da memória: onde serão colocados os programas e seus dados? Classicamente, temos duas opções.

No modelo Harvard, nomeado assim porque é como se fazia na Universidade de Harvard nos primórdios da computação digital, separa explicitamente a área de memória de programa (classicamente uma ROM, memória só de leitura) da área da memória de dados (RAM). Cada uma possui barramento exclusivo, idealmente. [Barramento ou bus é o conjunto de condutores (fios) por onde trafegam informações.]

`FIGURA: [Processador Harvard]`

Um exemplo bem comportado disso é o microcontrolador 8051, que possui instruções exclusivas para cada área: a ROM pode ser lida com MOVC (MOVe from Code memory), a RAM pode ser lida e escrita com MOVX (MOVe to/from eXternal memory). FIGURA: [Ele possui outras áreas de dados, além destas; microcontroladores modernos possuem mapas complexos de memória, com funções especiais para cada área disponível.]

Já o modelo von Neumann foi utilizado nos computadores que tiveram uma mãozinha de John von Neumann, uma das grandes figuras pioneiras da computação. Nestes, uma única memória contém tanto o programa quanto os dados necessários, acessados pelo mesmo barramento e mesmas instruções.

`FIGURA: [Processador von Neumann]`

Rigorosamente, podemos dizer que a distinção é: máquinas Harvard possuem espaços de endereçamento distintos para programa e dados, máquinas von Neumann possuem um espaço de endereçamento único. [Isso é lindo, até olharmos para microcontroladores modernos e, para não ficarmos desmotivados com a complexidade, decidirmos que esses conceitos não são tão importantes assim.]

A grande vantagem do modelo Harvard em relação ao von Neumann é que podemos realizar facilmente duas operações simultâneas, já que temos dois canais diferentes de transmissão de informação, e este recurso é amplamente usado nos processadores de alto desempenho atuais.

Já a vantagem do von Neumann diz respeito principalmente ao espaço físico utilizado (menos fios, menos CIs, menos controle necessário, menos projeto etc.) e à otimização do uso de memória. Seria estranho se fôssemos comprar dois pentes de RAM DDR para nossos PCs, sendo um para dados e outro para programas…

Haja vista o exposto, os computadores modernos são híbridos (podemos chamá-los de "Harvard Modificados"): possuem uma memória principal única (os pentes DDR), que como no von Neumann guardam tanto programas como dados, indistintamente, mas possuem uma memória cache bipartida internamente, como no Harvard, com uma área reservada para instruções dos programas e outra área para os dados acessados pelas instruções. Com isso, temos flexibilidade e desempenho juntos. [A memória cache é uma memória mais rápida colocada próxima ao processador para acelerar o desempenho e será vista oportunamente.]

RISC versus CISC: Decisões de Implementação

Estes conceitos são úteis apenas na medida em que nos possibilitam discutir escolhas de design para os microprocessadores, pois a classificação é um pouco ambígua quando colocada em prática.

RISC é um acrônimo que significa Reduced Instruction Set Computer. Leia bem: é um Computador que tem um Conjunto Reduzido (pequeno) de Instruções, não são as instruções que são pequenas, é a lista que tem poucas instruções. Estes processadores surgiram na década de 1980 quando alguns pesquisadores resolveram mensurar exatamente o que os processadores faziam quando estavam em funcionamento, e descobriram que algumas instruções eram usadas muitas e muitas vezes, enquanto outras apareciam muito raramente.

A ideia então foi simplificar o projeto do microprocessador, retirando instruções "exóticas" e deixando apenas o fundamental, para que com isso se ganhasse em desempenho: é melhor executar rápido as coisas comuns e demorar nas incomuns. Isso facilita muito a otimização dos circuitos, em especial abrindo a possibilidade de se utilizar a técnica de pipelining, presente em praticamente todos os processadores contemporâneos.

Com isto em mente, dois projetos foram criados nos EUA e financiados pela ARPA, a agência de fomento de pesquisa dos militares (a mesma que financiou a criação da Internet) [Sobre o fato da inovação verdadeira sair de fundos públicos e não de investidores privados, mesmo o Bill Gates admitiu isso recentemente. Pesquise vídeos do grande Noam Chomsky, o linguista do MIT que criou formalismos que usamos em compiladores e autômatos finitos, se quiser ver um discurso bastante esquerdista e coerente a respeito da transferência de fundos do setor público para o setor provado.] Um era o chamado Berkeley RISC, dirigido pelo Patterson do livro-texto; o outro era o Stanford MIPS, capitaneado pelo Hennessy do livro-texto. O MIPS, nosso objeto de estudo durante o semestre, foi líder de mercado na área de servidores durante os anos 90 e ainda hoje encontra aplicação nas áreas de telecomunicações e de processamento de sinais.

Como o Patterson criou o nome RISC, para diferenciar os processadores antigos destes novos ele criou o termo CISC: Complex Instruction Set Computers, ou seja, "o conjunto de instruções destes processadores é complicadão". Apenas numa primeira olhada, os processadores Intel atuais (Core i3, i5 e i7, da chamada arquitetura x86-64) e o processador dos mainframes IBM (System z13) possuem mais de 900 instruções cada, enquanto os RISC raramente passam de 100. Estes Intel e IBM são praticamente os únicos processadores CISC no mercado, além de alguns microcontroladores de 8 bits.

O paradigma RISC, que é praticamente impossível de ser seguido à risca, preza muito pela regularidade então. Por exemplo, as instruções RISC devem ter todas o mesmo tamanho (32 bits no MIPS) enquanto que as do CISC variam (podem ter de 1 a 17 bytes no x86-64!), o que provoca trabalho extra de decodificação.

Talvez o mais flagrante para o programador sejam as funções típicas de uma instrução: as RISC realizam uma tarefa pequena e simples ao invés de algo complexo e composto. Veja uma instrução típica do IA-32 (assembly 32 bits da Intel):

```
xor %eax,12(%ebx,%ecx,8)   # sintaxe AT&T, diferente da Intel (!)
```

Isso faz uma operação ou exclusivo entre o valor do registrador %eax e o conteúdo da memória na

posição dada por %ebx+%ecx*8+12; o resultado do XOR é gravado nesta mesma posição. O equivalente desta instrução em assembly MIPS32, usando $s0 para %eax e assim por diante, seriam cinco instruções:

```
sll $t0,$s1,3     # t0 <= s1<<3, ou seja, calcula ecx*8
add $t0,$t0,$s2   # t0 <= t0+s2, portanto t0 tem ebx+ecx*8
lw $t1,12($t0)    # t1 <= RAM[t0+12], lê da memória
xor $t1,$t1,$s0   # faz ou exclusivo com s0 (nosso eax)
sw $t1,12($t0)    # grava o resultado na RAM no mesmo lugar
```

Se quisermos piorar ainda mais a complexidade, o IA-32 possui a seguinte instrução:

```
REP MOVSB
```

Esta instrução copia um bloco inteiro da memória de uma localização para outra, podendo copiar até 64 kbytes utilizando milhares de ciclos de clock, numa instrução que só pode ser traduzida para o MIPS utilizando-se um loop, e isso pode ser bastante problemático para se otimizar num circuito da Intel. [Mas será que o REP MOVSB não fica mais rápido, já que o processador sabe que é um loop? "The "slowness" is only due to the lack of attention, by the processor engineers, to the REP; MOVS technique of moving data.", critica um sujeito qualquer em https://software.intel.com/en-us/forums/intel-visual-fortran-compiler-for-windows/topic/275765. Outro responde "On most of the newer CPU's, a simple loop moves memory faster than MOVS." e o engenheiro da Intel comenta embaixo: "As [the other answer] says, the simple loop, especially unrolled, is faster. Believe me, this is the sort of thing we pay VERY close attention to."] As instruções simples dos RISC, por outro lado, facilitam a implementação de técnicas avançadas de microarquitetura.

Além disso, os RISC acessam a memória de forma simples, usando uma instrução LOAD para ler um dado e uma instrução STORE para escrever um dado, sem realizar nenhuma outra operação. Os CISC típicos, por contraste, podem realizar múltiplos acessos de memória com uma única instrução, como no XOR exemplificado anteriormente.

Finalmente, as ISA RISC são ortogonais, o que significa dizer que uma instrução pode manipular qualquer um dos registradores disponíveis. [Ortogonalidade significa dizer que um operando não depende do outro — é ortogonal — em uma instrução.] Os CISC, tipicamente, possuem instruções que limitam o acesso a registradores específicos. Veja uma soma no microcontrolador 8051:

```
add A,R3
```

Esta instrução soma o registrador R3 ao registrador A e o resultado é gravado em A. Este registrador A é especial e é denominado acumulador nos CISC; todas as operações lógico aritméticas obrigatoriamente utilizam o acumulador como uma das fontes e como o destino da operação, ao contrário do MIPS, que pode somar quaisquer registradores e gravar o resultado em qualquer registrador.

```
FIGURA: [Tabela resumida comparando RISC e CISC]
```

Uma vez que RISC e CISC são apenas modelos idealizados, é discutível o quanto cada processador se aproxima do ideal. Em especial, os processadores ARM se dizem RISC mas possuem várias características bastante CISC, fugindo do modelo LOAD/STORE e executando mais de uma operação por instrução, como na instrução de transferência de dados entre vários registradores e

bloco de memória de uma vez só ou então no autoincremento de ponteiros. [Ou seja, um registrador usado no acesso à memória pode ter seu valor alterado logo antes ou logo depois da instrução, se o programador desejar.]

Modos de Endereçamento

Dada a importância dos acessos à memória, especialmente quando tínhamos poucos registradores, os processadores antigos desenvolveram esquemas sofisticados de operações com o objetivo de acelerar os programas (uma instrução complexa fazia mais coisas do que uma simples e executava mais rápido do que duas simples na era CISC). Então formas elaboradas foram desenvolvidas.

Os modos de endereçamento são as formas presentes no microprocessador para acessar dados para as operações. Por aqui, ficamos apenas com os modos presentes no MIPS, suficientes para nossa discussão, e listados abaixo:

1. Endereçamento imediato: o dado está presente como uma constante dentro da própria instrução. No MIPS, é algo como addi $s2,$s5,76 =⇒ a constante 76 está guardada dentro dos 32 bits do código de máquina da instrução.
2. Endereçamento via registrador: o dado está presente em um dos registradores internos, como em add $t5,$s2,$t0 =⇒ os três dados relevantes se referem a registradores.
3. Endereçamento direto: o dado está presente em memória e nós damos na instrução o endereço onde acessá-lo. A pseudoinstrução do MARS lw $s1,12004, por exemplo, lê o dado do endereço 12004, que é uma constate imediata.
4. Endereçamento indireto: é o exato equivalente do uso de ponteiros em linguagem C — os ponteiros são, de fato, traduzidos para instruções assim. Neste caso, um registrador contém o valor do endereço de memória onde o dado de interesse se encontra. Uma pseudoinstrução como lw $t3,($s6) ilustra o ponto: $s6 contém um número, e este número é o endereço de memória de onde vamos ler o dado, ou seja, $s6 é um ponteiro.
5. Endereçamento indireto indexado: similar à anterior, mas podemos indicar um "delta" de uma quantidade de endereços a ser somada/saltada. No formato nativo do MIPS, um sw $s0,2052($t3) vai armazenar o dado de $s0 no endereço obtido pela soma de $t3 com a constante 2052 (a constante é imediata, neste caso, está dentro da instrução em si).

Há outros modos mais complexos, mas a maioria dos processadores apresenta no mínimo os cinco modos listados acima. Algumas variantes simples podem ser interessantes, como endereçamento indireto indexado por registrador (no z80, por exemplo, temos LD (HL+IX),A que armazena A no endereço dado pela soma de dois registradores, o HL e o IX, criando possibilidades interessantes de uso) ou dados implícitos pela instrução (o topo da pilha). Endereçamentos indiretos com autoincremento e autodecremento dos ponteiros são populares, pois ajudam a construir pilhas, que são fundamentais para os códigos gerados por compiladores.

Na Intel, os vários modelos de organização de memória (em especial o obsoleto modo segmentado) dão origem a modos de endereçamento bastante desagradáveis de se aprender e fixar.

A Evolução do Microprocessador

Lei de Moore

Nenhuma discussão sobre a evolução dos computadores está completa sem a Lei de Moore. [Gordon Moore foi um dos fundadores da Intel, depois de trabalhar com o William Shockley.] Ela possui vários enunciados semelhantes, mas ficaremos com um dos mais comuns:

```
"A capacidade de colocarmos transistores num circuito integrado dobra a cada 18 meses."
```

Isso claramente implica num desempenho que aumenta exponencialmente e um custo por transistor que descresce exponencialmente, o que parece bastante anti-intuitivo para qualquer engenheiro. Surpreendentemente, a Lei de Moore continua sendo obedecida, de uma maneira ou de outra.

```
FIGURA: [Gráfico da Lei de Moore]
```

Largura de Palavra

Observamos o início da era dos microprocessadores (um processador colocado em uma única pastilha dentro de um Circuito Integrado) com o Intel 4004: um processador com largura de palavra de 4 bits criado para ser utilizado em calculadoras, que à época utilizavam código BCD em sua maioria.

A largura de palavra de um processador é, essencialmente, a quantidade de bits que a sua ULA (Unidade Lógica e Aritmética) utiliza nas suas operações. Por extensão, normalmente também é o número de bits armazenados em cada um dos registradores do processador.

Já em seguida, o 8008 aumentou para 8 bits a palavra, e o 8086 aumentou para 16. Há poucos anos, os processadores da Intel adotaram 64 bits. Quais as vantagens de se ter uma largura maior para a palavra?

Imediatamente, o tamanho dos números que podem ser expressos é menor com 8 bits: de 0 a 255. Ainda existem processadores de 8 bits no mercado: são microcontroladores pequenos e baratos, para tarefas simples, como acionamento de portões eletrônicos, controles remoto, fornos de micro-ondas. [Um microcontrolador é um circuito composto de um microprocessador associado a memórias de programa e dados, além de outros periféricos integrados como temporizadores e conversores A/D e D/A, tudo isto num único encapsulamento.] Exemplos são o AVR, usado no Arduino e RISC ainda por cima, e o 8051, um CISC muito usado em disciplinas de microcontroladores. Com o barateamento de outros processadores mais poderosos em versões simplificadas, em especial o ARM, eles estão perdendo terreno.

Fora dos PCs, estes ARM de 32 bits dominam o mercado, especialmente na telefonia celular e em sistemas embarcados. A expressão de mais números — de 0 a 2^32-1 — não só traz mais precisão e contas mais complexas, mas possibilita endereçarmos mais memória. Se tivermos um ponteiro de 32 bits, podemos endereçar 4 Gbytes de memória, em oposição a 64 kbytes se usarmos apenas 16 bits. [Aqui estou usando uma convenção simples: 1 kbyte = 1024 bytes, 1 Mbyte = 1024*1024 bytes e assim por diante. Existem convenções que tentam resolver esta ambiguidade de notação, mas nenhuma delas teve adoção ampla.]

As arquiteturas de 64 bits, como as dos Intel e AMD de nossos laptops mais recentes e de alguns

ARM, dobram o poder expressivo dos números. Ponteiros agora gastam 64 bits, portanto consomem 8 bytes para serem guardados numa variável (o que não é importante num PC mas este gasto extra pode fazer diferença num sistema embarcado), mas agora podem endereçar teoricamente até 16 exabytes (~16*10^18). [Por essa razão os Windows mais antigos, de 32 bits, tinham um limite de 4 GB de memória: qualquer quantidade adicional presente simplesmente não era utilizada.]

Como curiosidade, as arquiteturas antigas possuíam números diferentes de bits, não necessariamente múltiplos de 8 (veja na Wikipédia — em inglês — os DEC PDP com 12 ou 18 bits ou https://en.wikipedia.org/wiki/36-bit). Com o sucesso do IBM System/360 na década de 60, considerado um dos sistemas seminais, com mais influência na história, outros fabricantes adotaram o particionamento da informação em unidades de 8 bits, ao ponto que é raríssimo encontrar memórias que fogem do padrão de ter um byte por endereço.

Dependência de Caminho

Uma das curiosidades que pode ser observada facilmente no gráfico da Lei de Moore é a sequência de processadores criados pela Intel. Um dado um pouco assustador é que o código binário criado para rodar num processador 8088 de 1979 pode, em teoria, rodar sem alterações no Core i7 mais recente do mercado. Note que não é o programa em assembly, são os mesmos opcodes, que os processadores são obrigados a rodar sem alterações por uma questão de retrocompatibilidade: nenhum usuário gostaria de trocar de computador e ter de comprar novamente todos os programas que possuía por conta de uma mudança arquitetural.

Se considerarmos que o 8080 tinha um assembly parcialmente compatível com o do 8088, e que foi fortemente baseado em ideias vindas do 8008, que por sua vez foi uma expansão do 4004, podemos dizer que temos o DNA do primeiro microprocessador dentro dos nossos laptops…

Em respeito à Intel, eles tentaram por três vezes abandonar a arquitetura x86, que obviamente é inchada e cheia de características obsoletas que complicam desnecessariamente o circuito. Na primeira, criaram o ambicioso iAPX-432, um fracasso tão retumbante que foi expurgado da história oficial da Intel: com um assembly orientado a objetos e outras inovações interessantes, mostrou-se exageradamente lento. [Outros processadores exóticos existem, como os baseados em pilha e os assíncronos.] As outras duas foram melhores: os i960 de arquitetura RISC, que foram populares em sistemas embarcados (foram usados nas impressoras laser da HP por muito tempo, por exemplo) e o Itanium, com suas instruções VLIW nem RISC nem CISC (veremos mais à frente), que teve uma fatia não negligenciável do mercado de servidores e ainda é comercializado.

Se você está achando estranho o processador líder no segmento de computadores pessoais ter uma arquitetura CISC defasada, que dificulta otimizações e torna os circuitos mais lentos, tem razão. E dizer que o mercado e a retrocompatibilidade forçaram a mão da Intel é correto, mas não conta toda a história. (Aliás, a AMD também tem parte da responsabilidade nisso tudo, sendo efetivamente cocriadora dos CIs atuais).

O nome deste fenômeno é "dependência de caminho", e é estudado por economistas, que o definem aproximadamente como características de um produto sendo submetidas a externalidades do processo e não apenas à razão. E o exemplo padrão usado é o formato dos nossos teclados: o QWERTY.

Na segunda metade do século XIX, máquinas de escrever de vários fabricantes competiam no mercado americano, cada uma com seu layout de teclas diferente e acionamento particular. Ocorre que a mais popular se provou a Remington, com o design que conhecemos hoje, criado para se dificultar a digitação, pois o mecanismo da Remington fazia com que os martelos que imprimiam os tipos na fita se chocassem uns com os outros no caso de digitação muito veloz. Deixar as letras E e A, muito frequentes no inglês, a cargo dos dois dedos menos hábeis de uma pessoa destra, é apenas um dos mecanismos criados por eles para reduzir a rapidez.

Como na época havia mais gente que dominava o teclado QWERTY, mais cursos ofertados, mais máquinas disponíveis, novos usuários eram atraídos a esse padrão, já que retreinamento custa dinheiro e o domínio de padrões menos usados impacta na empregabilidade do funcionário e na dificuldade de achar mão de obra qualificada para o patrão.

Portanto, até hoje utilizamos um padrão meio arbitrário de teclas, criado há cerca de 150 anos, longe de ser o melhor possível, por pura e simples dependência de caminho: o padrão é definido pelo caminho histórico que trouxe o produto até aqui e não por características otimizadoras e sensatas, e não pode ser alterado sem grande custo.

O mesmo ocorre com os processadores Intel. O impacto é tão grande que, a certa altura, os engenheiros se viram obrigados a criar um outro assembly interno, RISC e invisível ao programador, que pode usar técnicas modernas de otimização, enquanto mantêm o assembly externo CISC, retrocompatível, sendo traduzido por circuitos extras para o interno.

Um exemplo mais divertido, porém não muito factual, é o do tamanho dos ônibus espaciais americanos, em especial a envergadura dos tanques sendo determinada pelo diâmetro dos túneis de trem, que dependeriam do transporte no império romano antigo.

Por que a Lei de Moore Funciona?

Pode ser surpreendente que uma lei, aparentemente de cunho prático, seja exponencial e ainda assim corresponda tão bem à realidade por mais de 40 anos. No mínimo, parece um otimismo exagerado.

A explicação, no entanto, é simples. O desenvolvimento de um novo microprocessador é bastante longo, levando entre 4 e 5 anos para uma nova geração de Intel Core, por exemplo, o que pode incluir mudanças na tecnologia de fabricação. Estimativas colocam o custo de uma nova fábrica, necessária quando muda o processo, quando há redução de tamanho, em mais de um bilhão de dólares, e são necessárias algumas fábricas, não uma só.

Portanto, quando o lançamento de um novo processador é planejado inicialmente, a empresa deve tentar adivinhar qual será o estado do mercado cinco anos à frente, algo impossível de ser feito. O resultado é que toda a indústria sempre se pautou na Lei de Moore para ter uma noção (os grandes players de setores tecnológicos usualmente produzem o chamado Roadmap da indústria, com as previsões para os próximos anos [http://www.semiconductors.org/document_library_sia/research_and_technology/international_technology_roadmap_for_semiconductors_itrs/?query=category.eq.Research%20and%20Technology&back=DocumentSIA]). Portanto tanto a Intel como seus concorrentes possuem previsões similares e metas similares, o que não impede que se cometam erros: uma das razões para

a AMD perder mercado atualmente foi chutar muito baixo (diga-se de passagem, as inovações da Intel também não são fáceis de competir). Em torno do ano 2000, foram as inovações da AMD que deixaram a Intel para trás. Mesmo o custo das memórias semicondutoras influencia neste processo: a destruição de uma fábrica importante no Sudeste Asiático forçou os preços acima e trouxe problemas de desempenho para usuários hesitantes em pagar mais caro.

Portanto, neste sentido, a Lei de Moore é uma "profecia autocumprida." Apesar disso, quanto mais perto chegamos dos limites físicos dos semicondutores, mais ameaçada está essa evolução. A Intel recentemente decidiu atrasar o lançamento de uma nova tecnologia de fabricação por conta dos problemas técnicos que a miniaturização está trazendo.

Consumo de Energia

As forças que impulsionam o desenvolvimento de novas tecnologias de microprocessadores são o aumento de desempenho e a diminuição do consumo. Este último é óbvio no nosso dia a dia, pois queremos que as baterias de celulares e laptops durem mais, mas tem também grandes consequências quando tomado em maior escala, especialmente em grandes data centers como os utilizados para armazenar dados e processar serviços, como os do Google.

Já o aumento de desempenho parece ser simples de entender: queremos que os celulares e PCs executem as tarefas mais rapidamente e que, com mais velocidade, novas tarefas, mais pesadas, se tornem possíveis. Mas temos a "Lei de Wirth", [O criador da linguagem Pascal, usada pelo Delphi] talvez um pouco irônica, dizendo que a complexidade do software mais do que duplica a cada 18 meses, tornando a Lei de Moore ineficaz.

De qualquer maneira, temos a questão: como aumentar o desempenho e diminuir o consumo?

Infelizmente, estas duas variáveis formam um trade-off, ou seja, um sistema de concessões mútuas: se uma melhora, a outra deve piorar e vice-versa. A razão para isso é que consumo maior e desempenho melhor são praticamente intercambiáveis, e a principal razão para isso é a tecnologia de fabricação de circuitos integrados. [Nota: além disso, permitindo-se gasto energético maior, pode-se fazer circuitos mais rápidos, independente da frequência]

Atualmente são utilizadas variantes da tecnologia CMOS (Complementary Metal Oxide Silicon) para construir os transistores que formam a base dos circuitos digitais: portas lógicas e células de memória. O consumo destes circuitos é dado por:

```
Pcmos = Pest + Pdin
```

onde Pest é a potência dissipada estaticamente, ou seja, quando o circuito está alimentado mas nenhuma alteração ocorre nele, e Pdin é a potência dinâmica, medida quando o circuito está em operação, mudando valores.

A potência estática consome cerca de 40% da potência nominal de um processador atualmente (2016), pois é essencialmente causada pelas correntes de fuga nos transistores, com portadores atravessando regiões que deveriam estar isoladas, com corrente igual a zero. Quanto menores os transistores, menor é a distância que o elétron precisa percorrer para "escapar" e portanto maiores essas correntes indesejadas.

Já a potência dinâmica é dada pela equação:

```
Pdin = V²*f*C
```

sendo V a tensão de operação, f a frequência de chaveamento e C a capacitância de junção. Várias observações importantes decorrem desta equação:

```
1) O consumo é proporcional à frequência
```

Isto se deve ao fato de que, num circuito CMOS, temos um FET de pull-up entre saída e alimentação e um FET de pull-down entre saída e massa. Como as impedâncias envolvidas são altíssimas, a corrente é praticamente zero quando o circuito está numa situação estável, consumindo apenas a potência estática devida às correntes de fuga. Porém, quando há um chaveamento, um dos FETs muda de corte para saturação e o outro faz o contrário simultaneamente; em determinado instante, ambos estão conduzindo parcialmente, com resistência de canal próxima do zero, o que provoca uma corrente significativa por um curto período de tempo. Portanto, quanto mais transições a porta lógica fizer, maior a potência dissipada.

```
FIGURA: [Porta Inversora CMOS]
```

É interessante notar que para obtermos maior desempenho de um microprocessador, basta aumentar a sua frequência de operação (caveat emptor: observe a Lei de Amdahl mais à frente), pois se dobrarmos o clock, teremos potencialmente o dobro de instruções sendo executadas num mesmo período de tempo. Assim, se o projeto do processador permitir aumentar a frequência de operação sem destruí-lo por excesso de temperatura, podemos aumentar seu desempenho facilmente.

```
2) Já que o termo da tensão em Pdin é ao quadrado, faz sentido investir antes de
mais nada na sua redução.
```

Isto explica como passamos dos 5V de alimentação nos anos 80 para cerca de 1,1V nos processadores modernos de alto desempenho. Abaixo deste valor, as tensões ficam difíceis de gerenciar pois as faixas de níveis lógicos ficam muito pequenas e os circuitos se tornam suscetíveis demais a ruídos.

```
3) A capacitância de junção C diminui conforme o processo de fabricação.
```

Quanto mais diminui o tamanho dos transistores MOSFET, menor é a largura do canal e tamanho do gate; portanto, a distância entre os contatos diminui, ou seja, temos menos material dielétrico e menor capacitância.

Assim, uma das vantagens da miniaturização é a diminuição proporcional da potência dinâmica. Portanto, nos interessa diminuir o máximo possível os transistores, embora isso aumente o consumo estático devido às correntes de fuga.

Tecnologias de Fabricação

As tecnologias de produção de circuitos integrados são usualmente expressas em nanômetros e, via de regra, o melhor da indústria é representado pelos processos empregados nos microprocessadores comerciais. Em 2016, a tecnologia de ponta é a de 14 nm; processos menores ainda estão sendo desenvolvidos em laboratório.

O que significa este número? Ele é definido como o "half-pitch" (meia-distância entre elementos idênticos numa matriz) para uma célula de memória. Uma célula é, essencialmente, um transistor e

um capacitor; se considerarmos ainda o espaçamento entre as células, esta distância de 14 nm é aproximadamente a largura de um bit, ou perto da largura de um transistor.

FIGURA: [Célula de Memória Dinâmica]

Se considerarmos a exigência prática da Lei de Moore, isso implica numa diminuição exponencial das dimensões envolvidas, o que se torna extremamente problemático: dentro de uma estrutura cristalina semicondutora, a distância entre o centro de dois átomos adjacentes de silício é quase 0,5 nm (os átomos em si possuem cerca de 0,1 nm de diâmetro); portanto a largura do canal dos FETs está em cerca de 30 átomos, atualmente.

Ao compreender o processo básico de fabricação, essa impressão piora: até hoje usamos variantes, mais avançadas, da fotolitografia utilizada nos primeiros CIs nos anos 70… Isso consiste em termos um substrato isolante na camada inferior, coberto por uma camada de material semicondutor ativo. Utiliza-se então uma máscara com o negativo do desenho do circuito, como as máscaras de PCB (printed circuit boards) ou de silkscreen para camisetas (serigrafia) e processos químicos para criar as junções semicondutoras. Todos os componentes ativos estão num mesmo plano, como numa folha de papel: a distribuição deles, portanto, é feita em apenas duas dimensões.

Por cima destas camadas que formam transistores, resistores e capacitores, são colocadas outras camadas bidimensionais com contatos entre si para fazer o roteamento das ligações, bem como os planos de alimentação e massa. Perceba que uma consequência deste arranjo é que a dissipação de calor vai seguir o módulo do circuito que está sendo usado, e a distribuição destes módulos é bidimensional, obviamente. [Estamos lentamente caminhando para 3D: a Intel usa, já desde a tecnologia de 22 nm, "transistores 3D", que na verdade continuam todos no mesmo plano, apenas possuem o gate empilhado em cima deles; fabricantes de FPGAs em 2010 começaram a fazer "empilhamento de pastilhas" — módulos com funções diferentes colocados em alturas variadas dentro do encapsulamento e interconectados por vias correndo as três dimensões; CIs verdadeiramente em 3D, no entanto, com um circuito sendo decomposto em várias alturas diferentes de transistores no mesmo encapsulamento, continuam sendo apenas tópico de pesquisa em laboratórios.]

Medindo Desempenho

Discutindo a relação da frequência de clock do microprocessador com o consumo, mencionamos de passagem que quanto maior o clock, mais instruções por segundo podemos executar. Podemos medir este valor em MIPS (Millions of Instructions per Second), mas algumas observações são pertinentes.

Em primeiro lugar, os conjuntos de instruções de processadores diferentes podem ser muito diferentes: comparar um MIPS com um Intel Core pode não fazer muito sentido, pois as instruções x86-64 são notavelmente mais complexas do que as RISC. Em segundo lugar, estes números são tipicamente valores de pico fornecidos pelos fabricantes, que obviamente escolhem a instrução mais rápida para a medição (sempre o NOP, "no operation", disponível em todo microprocessador): portanto, o número em si pode não ser muito representativo.

A comparação de desempenho entre dois sistemas é conhecida como benchmarking, nome derivado das hastes à beira de rios utilizadas para indicar qual o nível das águas atingido, em metros. O uso

de medidas relativas (comparando especificamente um sistema X com outro sistema Y) parece ser indicado, já que as medidas absolutas, como o MIPS, são questionáveis.

Uma alternativa é medir a quantidade de instruções usando programas específicos. O conjunto de testes conhecido por SPEC, por exemplo, usa programas reais que exigem bastante do processador, como compiladores, compressores de vídeo e programas para cálculos pesados de física e química. Um dos subtestes SPEC avalia a capacidade de um sistema completo de servidores de responder requisições externas, medindo transações por segundo. Já os testes CoreMark permitem a customização das tarefas a serem testadas, escolhidas dentre um rol de atividades comuns para sistemas embarcados (por exemplo, implementação de um pilha TCP/IP ou descompressão de vídeo h.264).

Apesar disso, duas medidas absolutas ainda são bastante utilizadas. FLOPS (Floating Point Operations per Second), usualmente medido em TFLOPS, que é bastante adequadas para aplicações científicas de "number crunching", essencialmente contas e mais contas com algoritmos numéricos (digamos, resolução de equações diferenciais para cálculo de reatores nucleares e coisas similares a isso). Dado este domínio, essa medida também é interessante na área de computação de alto desempenho (HPC - High Performance Computing), desde que seja uma medida em casos médios ao invés de pico. [Como trabalha-se com milhares, até centenas de milhares de processadores, a distribuição do workload é importante: as velocidades de transferência de dados nos barramentos e a taxa típica de ociosidade (que costuma ser alta) também são relevantes. Para informações sobre os maiores supercomputadores, cheque as listas de http://top500.org.]

A outra medida bastante utilizada, dentro do segmento de sistemas embarcados, é o Dhrystone MIPS, ou DMIPS. Ela indica a quantidade de instruções executadas por segundo, em milhões, para um teste específico: um programa dito sintético, consistindo de uma sequência de tarefas consideradas representativas de um hipotético "caso médio" para uma tarefa genérica, consistindo de contas, cópias de strings, loops e outras, chamado de benchmark Dhrystone. Infelizmente este caso médio não vai ser exatamente balanceado para todas as aplicações: um banco de dados usa muito o sistema de arquivos, uma simulação em física usa quase exclusivamente cálculos em ponto flutuante, algoritmos de busca farão muitas condicionais. O número final, portanto, serve apenas para dar uma ideia da capacidade do sistema em questão.

Por exemplo, um artigo de fabricante (Altera: http://www.atmel.com/images/mcu_vs_mpu_article.pdf), sugere que rodar um Android ou Linux embarcado exige um mínimo de 300 a 400 DMIPS; mas se um RTOS (Real Time Operating System), mais leve, for suficiente, cerca de 50 DMIPS já bastam. Se uma biblioteca gráfica for usada, 100 DMIPS adicionais sobre o Linux devem ser necessários no processador selecionado.

Tipicamente, as medidas são fornecidas em DMIPS @ Hz, como: "400 DMIPS @ 1 GHz". Isso ocorre porque o processador pode ser colocado para funcionar em uma gama de frequências, e por questões de economia de energia, talvez seja usado numa frequência muito inferior à frequência máxima. Os manuais, obviamente, trazem os números obtidos com o processador no talo das suas capacidades.

Cabe mencionar que há outros benchmarkings interessantes, incluindo alguns que medem indiretamente o impacto ambiental de Data Centers grandes, como os do Google, usualmente

construídos ao lado de rios para poderem utilizar a água para resfriamento das instalações e equipamentos (como nota ecológica, cabe ressaltar que a água aquecida pelo centro é transferida para piscinas antes de ser devolvida ao rio, para evitar impacto ambiental: a elevação de temperatura poderia ser prejudicial à vida local). Outros medem a eficiência energética de processadores, algo que é importante tanto para as baterias dos nossos celulares quanto na supercomputação: estudos indicam que para atingirmos a capacidade de exaFLOPS, uma pequena usina nuclear seria necessária para gerar a energia consumida pelos circuitos.

Power Wall

Retornando ao gráfico da Lei de Moore, uma coisa é notável no início dos anos 2000: o surgimento de chips multicore. Para simplificar, o número total de transistores continua seguindo a previsão, mas o desempenho de um núcleo único dentro de um microprocessador não continuou a escalada exponencial. Então temos dois, quatro, oito núcleos dentro de um mesmo encapsulamento, tipicamente com uma contagem de cerca de um bilhão de transistores (chegando até sete bilhões no caso extremo) em uma área usualmente entre 100 a 400 mm^2.

Mas por que razão mudar para multicore? O desempenho não poderia continuar sendo aumentado utilizando-se um processador único como anteriormente?

A resposta é não. Neste ponto da história, a indústria alcançou o limite prático de resfriamento para processadores comerciais, utilizando dissipadores metálicos e ventiladores potentes: se não for usada alguma técnica cara ou impraticável (nitrogênio líquido, digamos), processadores trabalhando em clock elevado vão aquecer por efeito Joule e a temperatura excessiva leva à ruptura mecânica da pastilha de silício. Portanto as frequências de clock usadas ficaram muito abaixo do que seria possível ou desejável.

Este momento de conversão para multicore é conhecido como Power Wall.

A partir de então, continua a evolução tanto da circuitaria para cada núcleo do processador quanto das frequências de clock, mas num passo bem mais modesto, fazendo com que a velocidade final do processador aumente pouco, em comparação com os aumentos escandalosos obtidos na década de 90. Atualmente, cada nova geração de processadores melhora apenas um pouco em cada área de interesse: clock, microarquitetura, consumo, taxa de transferência de dados, velocidade das memórias. Isso gera um aumento perceptível de velocidade de processamento, mas não um aumento brutal.

Tipicamente, o número de núcleos (microprocessadores independentes dentro de um mesmo encapsulamento) tende a aumentar no mercado destinado a consumidores caseiros: começamos com os dual core em torno de 2006, passamos a quadcores, hoje comuns, e temos octacores no topo de linha; servidores com 12 núcleos são comuns.

Os detalhes de Multithreading via Hardware serão vistos no final da matéria (esse é o Intel Hyperthreading), mas podemos adiantar que isso consiste em replicar parte do circuito de um processador para que ele aja como se fossem *dois*, simultaneamente. Com isso, podemos ter configurações como a do meu laptop: um Intel Core i5 com dois núcleos físicos (dois processadores de verdade) e quatro núcleos lógicos (os sistemas operacionais pensam que existem quatro processadores, e distribuem as tarefas pelos quatro, efetivamente). Isso gasta menos energia do que

quatro núcleos físicos e lógicos e apresenta melhor desempenho do que dois núcleos físicos e lógicos.

Exemplo de Mercado em 2016: *Portfolio* da Intel

Todo aluno de engenharia já teve que escolher um laptop para a mãe: a vida é o que ela é. Mas quem não é rato de internet (http://cpuboss.com, por exemplo?) pode ter tido bastante insegurança na compra. E com alguma razão. Alguém sabe a diferença de um Intel Core i3 para um i7? Não vale dizer "o i7 é mais rápido", apenas.

A linha da Intel desde 2015, a chamada 6ª geração, possui os Core i3, i5 e i7, que são melhores, além de Pentium e Celeron que têm menor desempenho, tanto em laptops quanto em desktops. No mercado mobile, de baixíssimo consumo, há todos estes e mais os Core m3, m5 e m7. Para servidores, temos apenas o Xeon E3. Processadores mais poderosos ainda não foram lançados, mas ainda são vendidos chips da 5ª geração; os da 6ª estão limitados, até agosto de 2016 ao menos, a 4 processadores físicos por CI, mas ainda assim parece haver uma grande variedade de opções para a esta 6ª geração.

Só que o significado de "6ª geração" é: "este processador possui microarquitetura codinome Skylake de 14nm". Se você lembra direito, isso significa que a organização da circuitaria interna de todos estes processadores é praticamente idêntica.

Olhando uma tabela com cuidado, vamos ver as diferenças essenciais entre os processadores Intel à venda: passam lentamente de processadores fracos mas bastante econômicos energética e financeiramente (i3) para melhor custo/consumo/benefício (i5) até chegar ao maior poder de processamento e gasto energético (i7), embora o circuito seja praticamente idêntico. [Na verdade, os módulos do processador — as caixinhas do diagrama em blocos — são mantidos, mas versões diferentes de implementação de cada um deles são usadas para balancear o trade-off entre desempenho e consumo]. A variação se dá, essencialmente, pelo número de processadores e threads, frequência dos clocks relevantes e quantidade de memória cache.

A história é: devemos confiar no trabalho dos marqueteiros da Intel, que sugerem os mercados-alvo dos i3, i5 e i7? Bem, sim e não (em ArqComp é sempre assim). Por um lado, tanto o *projeto* dos processadores quanto a *precificação* dos processadores têm estes mercados-alvo em vista, então se a empresa fez um bom trabalho, a proporção entre eles deve ser mantida.

No entanto, é comum a Intel chamar alguns processadores de laptop de "i7" mas eles terem um desempenho desapontador, pois como vimos antes, "Desempenho é consumo. Ambos são completamente intercambiáveis" (Patterson, numa entrevista). Para manter o laptop com alguma autonomia de bateria, ele tem que ser bem pior do que sua contraparte em desktop, onde podemos colocar fontes de centenas de Watts sem pestanejar. Observe os números para os processadores de 5ª geração (microarquitetura de codinome Broadwell: https://en.wikipedia.org/wiki/Broadwell_(microarchitecture) tem os números), em especial a quantidade de cores, clocks e gasto energético (TDP - Thermal Design Power).

Então a Intel ainda usa a microarquitetura antiga para cobrir os mercados que a Skylake não teve tempo de entrar, como os Core i7 Extreme octacore de 16 threads (16 processadores lógicos, como visto na seção anterior) e os Xeon de dezenas de núcleos para servidores (além do Itanium, que é

um processador não derivado do x86). Há ainda outros mercados, como os sistemas embarcados, no qual a placa Galileo com o Intel Quark visa competir com o Arduino e o Raspberry Pi; a computação científica "caseira" (barata e de pequena escala mas alto desempenho), com o Xeon Phi de até 72 cores que evita a desagradabilíssima tarefa de programar a placa de vídeo para isso, algo bem comum; e processadores x86 acoplados a FPGAs (circuitos digitais programáveis) num mesmo encapsulamento para algumas aplicações embarcadas de alto desempenho, graças à compra da Altera pela Intel, que sempre dividiu o mercado apenas com a Xilinx nesta área.

Lei de Amdahl

Se o desempenho de um sistema depende de tantas variáveis, como velocidades do processador, memória, HD, etc., qual o impacto global de uma melhoria feita em apenas uma destas variáveis? É disto que trata a Lei de Amdahl (do Gene Amdahl, arquiteto chefe do importantíssimo IBM System/360), relacionada à Lei dos Retornos Decrescentes estudada em Economia.

Digamos assim: vamos supor simplificadamente que um sistema, chamado Chapolin, depende essencialmente daquelas três características listadas anteriormente: clock, taxa de transferência máxima de dados da RAM e velocidade de acesso do disco rígido (isso é mais ou menos verdade). Aí vamos supor que cada uma destas variáveis contribui com 1/3 do impacto no processamento, ou, de outra forma, se levamos 3s para executar um programa, o processador gasta 1s, o HD 1s e a RAM 1s, considerando operações sequenciais.

Se um engenheiro meio míope decide investir todos os esforços em apenas uma das variáveis, qual será o ganho? Se ele *dobrar* a velocidade da RAM num sistema aperfeiçoado chamado de DidiMocó, o tempo total de processamento será 2,5s: um ganho de 0,5s/3,0s = 16,67%. Se ele *quadruplicar* a velocidade da RAM, o ganho em relação ao sistema inicial Chapolin será 0,75s/3,0s = 25%; o ganho em relação ao DidiMocó será de 0,25s/2,5s = 10% apenas. Cada vez que ele dobra a velocidade da RAM, ele obtém um ganho menor em relação ao sistema global (considerando as demais variáveis inalteradas).

No limite, se o engenheiro Mr John Super Optimizer conseguir fazer uma RAM com velocidade infinita, depois de alguns anos e muitos milhões de dólares, o sistema geral ainda vai gastar 2s para executar aquele programa inicial que nos interessava, uma melhora de 33% apenas. Portanto há um limite claro da contribuição que a otimização pode trazer, e esta é uma das consequências interessantes da Lei de Amdahl, que podemos enunciar agora:

```
"A quantidade de aceleração que podemos obter num sistema computacional ao otimizarmos uma característica específica é proporcional ao quanto esta característica influi no sistema como um todo."
```

O enunciado fica um pouco confuso mas é mais ou menos assim mesmo. Já me conformei com isso.

Outra consequência interessante diz respeito à escolha do que deve ser otimizado: essencialmente, devemos nos concentrar fortemente nos gargalos de desempenho do sistema, pois aí é onde estão os maiores ganhos potenciais. Uma melhora boa no disco rígido, de longe o componente mais lento no sistema (veremos números quando estudarmos memórias), tem um grande impacto no sistema — não à toa todos os computadores da Apple utilizam SSDs (pra quem não sabe: memórias Flash RAM não voláteis que subsituem o disco magnético no papel de HD com muita velocidade). Por outro lado, melhoras na velocidade de execução de um único processador ajudam para algumas

tarefas, mas em geral não são tão perceptíveis no desempenho geral.

Também nos interessa a ideia de paralelizar a execução de programas. Um primeiro aspecto é o quanto é paralelizável: para qualquer problema, temos uma parte dele que é primariamente sequencial e não é paralelizada. Números são gostosos, então vamos inventar uns: um problema leva cem mil minutos para executar com um único processador e, viva!, só 1% dele não é divisível entre vários processadores! Então usamos cem mil de processadores (isso existe, ok?). Só que aquele 1% representa mil minutos não paralelizáveis, então mesmo se tivéssemos um bilhão de processadores não conseguiríamos baixar de mil minutos.

Um segundo aspecto é qual o ganho que podemos obter com a paralelização. Um exemplo prático é a compilação paralela: se formos compilar um projeto grande em C com o make, temos a opção de dividir a execução em vários processadores criando, por exemplo, 8 jobs (8 processos paralelos, essencialmente) com a flag -j8. Haverá ganho? Como de hábito, a resposta é "depende."

A maioria dos processos de compilação é "I/O bound", ou seja, limitada pelo acesso intensivo a disco; neste caso, haverá algum refresco porque quando um processador estiver fritando com um arquivo, outro processador pode estar lendo o próximo. Como este tempo de fritação é curto, o ganho provavelmente não será tão significativo.

Por outro lado, a compilação pode ser "CPU bound", ou limitada pela quantidade de processamento, embora isso seja infrequente. Neste caso, veremos um speedup considerável no tempo total de compilação. [Veja discussões como esta: http://stackoverflow.com/questions/414714/compiling-with-g-using-multiple-cores]

Por fim, nesta era de processamento de quantidades massivas de dados, um dos gargalos mais notáveis, tanto no mercado de consumidores caseiros quanto de alto desempenho, é a banda de memória, ou seja, a taxa de transferência de dados pelos canais de comunicação em torno do processador. Observe que a Intel sempre faz questão de dar números para seus barramentos principais quando lança um novo processador, e que os supercomputadores dão bastante destaque às tecnologias de barramento utilizadas na topologia escolhida.

Mas Afinal, É Vantagem Ter Tantos Processadores?

Sim e não. Isso depende essencialmente do tipo de aplicação que está sendo usada no hardware, e de quantas coisas simultâneas desejamos e podemos realizar.

Se temos um computador para realizar apenas uma tarefa (o Word ou o Campo Minado), não há razão para ter mais de um núcleo. Se fazemos várias coisas simultaneamente, em especial coisas pesadas (tratamento de fotografias, compressão de vídeo e compilação de qualquer coisa usando o Eclipse), podemos ter bastante vantagem nisso, pois o sistema operacional automaticamente irá encaminhar novas tarefas para processadores que se encontram livres. (Observe, por obséquio, sem obsessão, a taxa de utilização dos seus processadores utilizando o Monitor de Sistema do seu Linux — ou Windows, caso você seja um destes traidores, ou OS X, se você for um programador gourmet. Caso a taxa seja baixa em todos os processadores — sei lá, 25%? — tem poder sobrando; caso esteja alta — aí é tipo 60% pra cima — eles estão no talo e aumentar seu número pode ser benéfico).

Claro que se a única tarefa importante for pesadíssima, como renderização de vídeos de computação gráfica (é isso mesmo, os dinossauros do Jurassic Park), é bom ter pencas de processadores, com

uma pegadinha: o problema tem que ser divisível graciosamente, digamos.

Num extremo, temos casos como a alteração de brilho de uma fotografia no Photoshop. Basta recortarmos a fotografia em, por exemplo, quatro pedaços, mandar um para cada processador, deixá-los fazer o ajuste, e finalmente colar os pedaços de novo na tela. Isso ocorre porque a ação é simples: cada ponto tem seu valor RGB multiplicado apenas (multiplique a quantia numérica de vermelho, verde e azul por 1,2 e terá um aumento de 20% no brilho) e mais nada, portanto um ponto não depende de outro e pode ser processado sem considerar seus vizinhos.

Este tipo de problema é chamado de embaraçosamente paralelo.

No outro extremo, temos casos como o cálculo das casas decimais do número irracional pi. Para o cálculo de cada casa, todas as casas anteriores devem ter sido calculadas e portanto não conseguimos dividir o trabalho entre diferentes processadores… Assim, todo o trabalho deve obrigatoriamente ser feito em um único processador (curiosamente, podemos descobrir uma casa *binária* de pi sem precisar das anteriores, o que não ajuda nosso problema inicial). Este tipo de problema diz-se inerentemente sequencial.

Mais sobre isso nas aulas de multiprocessamento ao final do semestre. Mas dá pra passar duas disciplinas inteiras falando sobre isso, se for o caso.

Alternativas?

No mercado sanguinário de tecnologia, as startups do Vale do Silício estão todas com faca nos dentes, então não faltam opções excêntricas desenvolvidas, às vezes do dia para a noite, às vezes ao longo de décadas.

A tecnologia celebrada como "mais promissora" atualmente é a computação quântica. Ainda não dispomos, em 2016, de um verdadeiro computador quântico genérico, mas os investimentos em pesquisa são intensos e avanços são constantes. Tem um adendo sobre isso ao final, pra quem quiser ler. Existem outras tecnologias correndo pelas beiradas, como a computação fotônica.

Circuitos assíncronos proveem uma alternativa já conhecida; um processador assíncrono com 144 cores já foi feito e comercializado (procure pela Green Arrays Inc.), mas sem grande sucesso.

As mudanças em tecnologia de fabricação também dão algumas esperanças de desenvolvimento; em especial, a fabricação em 3D "verdadeiro" (já se usa o que se chama de 2.5D, que são pastilhas empilhadas e não um circuito único roteado tridimensionalmente).

Na arena de multiprocessamento, utilizar placas de vídeos poderosas para realizar cálculos em conjunto com a CPU se tornou comum (GPGPU — General Processing on Graphics Processing Unit), inclusive sendo uma ameaça para soluções de criptografia pouco poderosas. Nos sistemas embarcados, começa a ficar cada vez mais popular a existência de sistemas heterogêneos, nos quais processadores diferentes entre si executam tarefas especializadas de forma coordenada; podemos, por exemplo, rodar um Linux num processador para as tarefas gerais, rodar um sistema de tempo real em outro para o controle de tarefas importantes/inadiáveis e usar uma FPGA ou uma GPU para cálculos pesados, tudo no mesmo sistema.

Adendo Sobre Aritmética: Complemento de 2 e Ponto Flutuante

Complemento de 2 foi visto em sala, embora eu tenha esquecido de falar a diferença entre carry e overflow. Ponto flutuante eu acabei não passando. Eu se fosse vocês dava uma estudada, porque é importante, mas só vou colocar na prova do 2º sem. 2016, se eu for colocar, algo daquilo que foi passado no quadro.

Sorry pela falta desta seção.

Tópicos Interessantes: Motivação para Perder Tempo na Web

É vendo ideias diferentes que estimulamos nossa criatividade e imaginação, atributos valorizados hoje em dia no meio tecnológico por gerar inovação. Ademais, elas nos fazem refletir sobre as premissas que assumimos por default, suas fundações e consequências: portanto nos fazem compreender melhor os conceitos que utilizamos. Então, faltando os links ainda:

- Há a possibilidade de fazer circuitos assíncronos (sem clock para sincronizá-los), com a vantagem de não haver desperdício de tempo de espera. Como isso pode ser feito? LINK
- Quando se faz sistemas multiprocessados massivos (supercomputadores) a distribuição dos dados se torna muito importante, assim como cada nó da rede, seja ele uma CPU multicore ou uma GPU. Quais as velocidades de transferência típicas? Quais as topologias de ligação? LINK
- Há vários exemplos históricos de tecnologias excêntricas, como o computador ternário (tinha que ser russo; como construir um computador ternário? LINK), Stack Machines (LINK), sistemas OISC e ZISC (LINK) etc. (Sim, eu adoro a Wikipédia inglesa).
- Uma alternativa proposta ao uso de pipelines superescalares é chamada Belt Machine; nela, o valore de um registrador passa para o registrador seguinte a cada clock (o R1 passa para o R2, que passa para o R3 e assim por diante), de forma que qualquer variável vai caminhar entre os registradores sem um número fixo. Como programar algo assim? LINK
- Como é o processo de fabricação de um CI? LINK
- Como a IBM conseguiu manipular átomos individualmente para escrever a palavra "IBM" usando um microscópio de tunelamento? LINK
- Os raios cósmicos são partículas provenientes do espaço que são em boa parte absorvidos pela atmosfera. No entanto, ensaios foram anteriormente conduzidos em cavernas dezenas de metros abaixo da superfície para garantir que eles não provocassem erros nos Circuitos Integrados. Estes erros foram minimizados ao ponto de não nos preocuparmos mais com isso. Como? LINK
- A história da computação também é cheia de coisas bacanas, desde a Máquina de Anticítera — que é um computador analógico mecânico que determina movimentos astronômicos há dois mil anos — passando por Ada Lovelace, a primeira programadora e por Konrad Zuse, inventando o primeiro computador programável na Alemanha Nazista na mesma época em que o Ocidente criava o Bombe em Bletchley Park para quebrar a criptografia do Enigma (O QUÊ? VOCÊ NÃO VIU "THE IMITATION GAME", sobre o gênio da computação Alan

Turing? VÁ JÁ PARA O NETFLIX).

- Representação numérica também é massa. O número 1/3 é impossível de representar em base decimal sem infinitos dígitos ou arredondamento… Vamos mostrar:
    - 1/3 = 0,333… (os três pontos significam infinitos dígitos decimais 3)
    - Com dois dígitos significativos, 1/3 ≈ 0,33 (portanto o erro é: 0,333…-0,33 = 0,00333…)
    - Com quatro dígitos significativos, 1/3 ≈ 0,3333 (portanto o erro é: 0,333…-0,3333 = 0,0000333…)
    - Com sete dígitos significativos, 1/3 ≈ 0,3333333 (portanto o erro é: 0,333…-0,3333333 = 0,0000000333…)
    - Portanto é impossível representar o número 1/3 em decimal com uma quantia exata…
    - No entanto, 1/3 em base 3 é apenas 0,1(3). Esta quantia é EXATA, e possui apenas um dígito ternário (!)
    - Portanto a precisão e exatidão de um número depende da base em que ele é expresso.
    - Em algumas aplicações, é necessário ter precisão arbitrária (número de casas decimais escolhido pelo programador) e em outras queremos ter números exatos (números racionais). Como eles são representados e manipulados? LINK
    - E existem, claro, opções excêntricas, como expressar números em uma base negativa, irracional ou complexa (that damned ducking Wikipedia again - LINK), por exemplo.
    - E se formos usar um sistema não posicional, podemos usar o peso de cada dígito binário como números consecutivos da série de Fibonacci. Como funciona isso? LINK. A representação resultante é um exemplo de código autossincronizável, ou seja, se perdermos um bit de dados numa transmissão, perderemos no máximo dois números e o fluxo voltará ao normal.
- Foi mencionado brevemente no texto que para atingir exaFLOPS seria necessária uma pequena usina nuclear apenas para fornecer energia para os circuitos. Este estudo foi encomendado pelo Exército Americano (LINK). Reflita sobre as condições geopolíticas que levaram à encomenda deste estudo.
- Memórias falham, especialmente devido às minúsculas dimensões em que são fabricadas atualmente (digamos, simplificadamente, que é comum haver um erro pequeno no processo de litografia — basta estragar um bit, afinal… — ou um ruído pontual, ou uma pequena flutuação eletromagnética indesejada). Para aplicações de alta confiabilidade, como alguns servidores, códigos de correção de erro são utilizados, como o CRC. Essencialmente, criamos uma redundância, mapeando três bits em cinco, por exemplo, de forma que podemos facilmente corrigir um erro que ocorra em um único bit e garantir a detecção de

erros se ocorrerem em até dois dos cinco bits (!). Como funciona isso? Qual o overhead de dados? Quando é adequado ou desejado usar? LINK INICIAL

- Processadores falham, e quando eles estão controlando aviões ou usinas nucleares, um mero bit que seja invertido indesejadamente pode ser catastrófico. A solução típica nestes casos é oferecer redundância. Num dos arranjos possíveis, três processadores fazem a mesma coisa e "votam": se algum deles discordar, é descartado como defeituoso. Em outro, um processador executa a mesma coisa que o principal mas com meio clock de atraso (pois os ruídos eletromagnéticos costumam ser bem rápidos), e caso haja discrepância, o programa é reexecutado a partir do último ponto correto. LINK

Um Pequeno Adendo sobre Computação Quântica

As introduções a computação quântica que vi por aí são péssimas: ou são muito genéricas, pra estudante de ensino médio (portanto inúteis pois não ajudam a entender) ou são muito técnicas, pra doutorando em física (portanto inúteis para nós). Vou tentar um resumidão. Perdoem os erros conceituais.

Utilizando alguns chunchos quânticos (isso não é jargão técnico, note-se), um processador quântico utiliza qubits ou quantum bits ao invés de bits ordinários de informação. Um qubit pode estar em estado 0 ou em estado 1 simultaneamente; lembre lá do gato de Schrödinger: antes de abrir a caixa que contém um gato e um veneno que pode ou não ser liberado, não sabemos o que aconteceu, então o gato está vivo e morto ao mesmo tempo [Note que não é que ele está em um dos dois estados lá dentro e a gente só não sabe: ele está nos dois estados, simultaneamente vivo e morto — essa história é uma metáfora e não uma explicação, é uma analogia inexata]. Isto se chama superposição de estados. Então se tivermos um número n de 3 qubits, teremos:

```
0/1 0/1 0/1
```

O que potencialmente representa oito estados clássicos: os bits 000, 001, 010, 011, 100, 101, 110 e 111. Isso é bastante interessante para fazermos, por exemplo, uma busca: ao invés de fazer um loop elencando as oito opções, simplesmente temos n, que simultaneamente representa as oito opções, usando a superposição de estados quânticos.

O truque é achar qual das 8 opções é a que buscamos (isso é, efetivamente, o centro do algoritmo quântico): fazemos o número n colapsar seu estado superposto para um dos 8 estados possíveis, tentando obter a resposta que procuramos, o que chamamos de decoerência. Imagine o uso disso para uma chave assimétrica em criptografia, por exemplo (se não sabe o que é isso, dá uma olhada): precisamos descobrir dois números primos muito muito grandes que multiplicados geram um outro número, e a busca é muito demorada, mas com dois "números quânticos" poderíamos em tese descobrir os primos em apenas um passo!

O problema técnico essencial é fazer funcionar bem esse passo, pois ele depende do sistema ser adiabático, hermético, o que é impossível, portanto temos erros: o resultado final é correto com alguma probabilidade maior do que 50%, que pode ser calculada. Portanto um algoritmo quântico é um algoritmo probabilístico, não determinístico. Livros e livros já foram escritos apenas sobre algoritmos probabilísticos, que são práticos e usados frequentemente. Mais outra montanha foi escrita sobre os quânticos, que continuam sendo teóricos.

Até 2016, o cálculo determinístico mais complexo já demonstrado foi a fatorização do número 143 usando 4 qubits. Mas depois que a NASA e o Google compraram computadores quânticos de 512 qubits da D-Wave para pesquisa, vários pesquisadores céticos começaram a levar o potencial da área mais a sério. Claro que, no caso do Google que realiza Deep Learning, essencialmente um procedimento estatístico com resultado probabilístico onde erros são parte do processo, o *quanto* isso realmente está funcionando não importa muito. Ainda há controvérsias se este processador realmente apresenta os efeitos quânticos que o fabricante afirma (é impossível de se medir com certeza, embora haja indícios encorajadores) e ele não é um computador quântico genérico, mas realiza Quantum annealing (arrefecimento quântico), uma subclasse. E, claro, funciona a temperaturas próximas de zero Kelvin.

E mesmo que a computação quântica realize completamente o prometido, isso não significa o fim da criptografia (já existem procedimentos bem definidos de criptografia quântica para quando a tecnologia chegar lá — veja por exemplo https://en.wikipedia.org/wiki/Quantum_key_distribution e https://en.wikipedia.org/wiki/Quantum_Experiments_at_Space_Scale). Um processador quântico não consegue fazer uma redução exponencial na complexidade de qualquer problema, como pode parecer à primeira vista. Os estudos indicam que há sim uma classe de problemas difíceis que podem ser tornados fáceis com ele, mas não todos: em particular, não se sabe se ele pode resolver eficientemente os problemas NP completos (sabe-se que problemas resolvidos eficientemente por computadores quânticos estão em PSPACE e encampam P; suspeita-se que não encampem NP completos mas que incluam parte de NP). [Se você é de Eletrônica ou fugiu de Teoria da Computação, vamos dizer que um problema NP completo é um problema bem difícil, de uso prático, que quebra as pernas de um computador tradicional e exige muita acrobacia e pirotecnia para se obter uma solução, seja aproximada ou não. Já P são os problemas com solução em tempo polinomial, ou seja, problemas molinhos de resolver].

Last updated 2016-09-17 17:32:41 BRT