Sumário

- Limitação da coerência sequencial
- Modelos de coerência enfraquecida.

Problemas da coerência sequencial

- Qualquer concretização é limitada pela necessidade de estabelecer uma ordem total nas operações de escrita.

  Isto é inerentemente dispendioso num sistema distribuído.

- Por isso têm sido procurados modelos de memória mais fracos.

  Mas que podem ser concretizados de modo mais eficiente.

Memória causal

- Operações de escrita (potencialmente) relacionadas de modo causal, devem ser vistas pela mesma ordem. Escritas concorrentes podem ser vistas por ordem diferente.

Memória causal: história inválida

P_1	$W_1(x)1$		
P_2		$R_2(x)1$ $W_2(x)2$	
P_3			$R_3(x)1$ $R_3(x)2$
P_4			$R_4(x)2$ $R_4(x)1$ (oops)

Memória causal: história válida

P_1	$W_1(x)1$			
P_2		$W_2(x)2$		
P_3			$R_3(x)1$ $R_3(x)2$	
P_4			$R_4(x)2$ $R_4(x)1$	(não há relação de causalidade)

Programando com memória causal

- Pretende-se resolver a seguinte equação $Ax = b$.
- Usando um algoritmo de iterativo.

$$x_i^{k+1} = (b_i - \sum_{j=1}^{i-1} a_{i,j} x_j^k - \sum_{j=i+1}^{n} a_{i,j} x_j^k)/a_{i,i}$$

- Para calcular x_i^{k+1} é necessário ver todas as actualizações de x_i^k da iteracção anterior.
- Cada elemento da solução pode ser calculado num nó diferente (processamento paralelo).

Solução usando memória causal

– cordenador

while ($\neg$ done)
    wait until ($\forall_i$ complete$_i$)
    foreach i
        complete$_i$:= false
    wait until ($\forall_i$ changed$_i$)
    if (converged())
        done := true
    foreach i
        changed$_i$:= false

– trabalhador

while ($\neg$ done)
    $t_i = (b_i - \sum_{j=1}^{i-1} a_{i,j} x_j - \sum_{j=i+1}^{n} a_{i,j} x_j)/a_{i,i}$
    complete$_i$:= true
    wait until ($\neg$ complete$_i$)
    $x_i := t_i$
    changed$_i$:= true
    wait until ($\neg$ changed$_i$)

Porquê funciona?

- A solução anterior funciona porque o coordenador estabelece um elo causal entre os vários trabalhadores.

  Cada trabalhador actualiza x_i antes de escrever `changed`$_i$ `:= true`.

  O coordenador espera por todos os `changed`$_i$ `:= true` *antes de voltar a escrever* `changed`$_i$ `:= false`. *Logo, esta última actualização depende causalmente de todas as actualizações de x_i.*

  Cada trabalhador lê `changed`$_i$ `:= false`. *Logo, os valores de x_i são propagados por essa dependência causal.*

PRAM

- Escritas por um mesmo processo são vistas pela ordem com que são especificadas, mas não se ordenam escritas de processos diferentes (mesmo que relacionadas causalmente).

P_1	$W_1(x)1$
P_2	$R_2(x)1\ W_2(x)2$
P_3	$R_3(x)1\ R_3(x)2$
P_4	$R_4(x)2\ R_4(x)1$ (legal apenas na PRAM)

Modelos baseados em sincronização

- Na maioria das aplicações paralelas, o acesso a memória partilhada está associado à utilização de primitivas de sincronização (ex: semáforos de exclusão mútua).
- Uma optimização do sistema, consiste em garantir a coerência da memória apenas nos momentos em que se estabelece a sincronização.
- Sem se tentar manter a coerência da memória nos restantes intervalos.

  Dado que não é suposto haver acessos concorrentes sem passar pela sincronização.

Modelos de coerência fraca

- Os modelos baseados em sincronização são também chamados de modelos de coerência fraca:

  Porque não oferecem garantias caso o programa não utilize correctamente as primitivas de sincronização.

- No entanto, na prática, o objectivo é fornecer uma memória sequêncialmente coerente.

  A todos os programas que usem correctamente as primitivas de sincronização.

Coerência fraca (Dubois et al 1986)

- O acesso a variáveis de sincronização é sequencialmente coerente.
- Os acessos às restantes variáveis são ordenados de modo total em relação às operações de sincronização.

P_1	$W_1(x)1\ W_1(x)2\ \mathbf{S}\ R_1(x)2$
P_2	$R_2(x)1\ R_2(x)2\ \mathbf{S}\ R_2(x)2$
P_3	$R_3(x)2\ R_3(x)1\ \mathbf{S}\ R_2(x)2$
P_4	$R_3(x)2\ R_3(x)1\ \mathbf{S}\ R_2(x)1$ (oops)

Coerência na saída (release consistency)

- Semelhante ao anterior mas distinguem-se as operações:

  Aquisição: para aceder aos dados.

  Libertação: para libertar os dados.

- Uma cópia actualizada dos dados é obtida no aquisição e disponibilizada na libertação.

Coerência na entrada (entry consistency)

- Semelhante à coerência na saída:

  Cada variável deve estar associada a um objecto de sincronização próprio.

  Fácilmente aplicável a modelos orientados aos objectos, em que se pode associar uma variável de sincronização a cada objecto.

Concretização de modelos fracos

- Semelhante à memória atómica embora:

  As actualizações não são feitas a cada acesso à página.

  Mas apenas quando se executa uma acção de sincronização.

  Quatro tipos de apróximação: propagação/invalidação imediata ou propagação/invalidação diferida das alterações (lazy).

Coerência na saída com propagação imediata

- Quando se executa uma operação de Libertação propagam-se as actualizações/ invalidações.

- Nenhum outro nó pode realizar uma operação de Aquisição enquanto estas actualizações/ invalidações não estiverem confirmadas.

  Geralmente as actualizações são enviadas na forma de diffs, isto é, são apenas enviadas as alterações e não uma nova cópia da página.

Coerência na saída com propagação diferida (lazy)

- Quando se executa uma operação de Libertação não se propagam as actualizações/ invalidações de imediato.
- Quando um nó realiza uma operação de Aquisição recebe todas as actualizações (ou invalidações) feitas no passado (definido como uma ordem causal).

Resumo

- Memória causal.
- PRAM.
- Release consistency.
- Lazy release consistency.