EXPERIMENTOS COM BARREIRAS EM MÁQUINAS PARALELAS

JAIRO PANETTA

INSTITUTO DE ESTUDOS AVANÇADOS

CENTRO TÉCNICO AEROESPACIAL

SUMÁRIO

Apresenta-se um algoritmo paralelo para a sincronização de n processos usando barreiras. O algoritmo tem complexidade O(log n) para um numero de processadores maior ou igual a n. Apresentam-se resultados experimentais que são comparados com uma implementação sequencial.

1 - INTRODUÇÃO

Barreiras são construções de software para a sincronização de processos. Uma barreira para n processos retém os primeiros n-1 processos que a atingem até que o último processo se apresente. Neste evento, todos os n processos estão sincronizados e são liberados para prosseguir sua computação.

Barreiras são usualmente implementadas por duas operações: uma que estabelece a capacidade da barreira (número de processos a sincronizar) e outra que sincroniza os processos, retendo-os até que todos se apresentem. Sejam defina_barreira (n, nome) e espera_barreira (nome) tais operações. Para exemplificar seu uso, considere uma computação iterativa onde um processo mestre envia trabalho para n-1 processos escravos e para ele mesmo, após verificar os resultados dos n processos na iteração anterior. Observe que há dois pontos de

sincronização: o mestre enviará trabalho após todos os escravos terminarem a iteração anterior, e os escravos iniciarão a próxima iteração após receberem o trabalho a desenvolver. Uma implementação típica utiliza duas barreiras, uma em que os escravos aguardam o envio de trabalho pelo controlador (barreira mestre) e outra em que o controlador aguarda o término do trabalho de todos (barreira escravo), utilizadas na forma a seguir

MESTRE	ESCRAVO
defina_barreira (n,mestre);	
defina_barreira (n,escravo);	
crie n-1 processos escravos;	
repita k vezes	repita k vezes
envie trabalho;	
espere_barreira (mestre);	espere_barreira (mestre)
realize trabalho;	realize trabalho;
espere_barreira (escravo);	espere-barreira (escravo)

O custo de sincronização pode ser diminuido (elimando-se a barreira mestre) se a semântica da barreira for alterada. Suponha que após todos os processos atingirem a barreira, apenas um deles (privilegiado) seja liberado. Todos os outros aguardam na barreira até que o processo privilegiado envie uma ordem para liberá-los. Os processos são numerados de 0 a n-1, o que permite distinguir o processo privilegiado (digamos, 0) dos outros. As operações passam a ser defina_barreira (n,nome), espera_barreira (id,nome) e abra_barreira (nome). O exemplo anterior passa a ser implementado por

```
MESTRE                                ESCRAVO NÚMERO p

defina_barreira (n,escravo);
crie n-1 processos escravos;
repita k vezes                        repita k vezes
  espere_barreira (0,escravo);          espere_barreira (p,escravo);
  envie trabalho;
  abra_barreira (escravo);
  realize trabalho;                     realize trabalho;
```

2 - IMPLEMENTAÇÃO SEQUENCIAL DE BARREIRAS

Uma implementação comum para barreiras com a primeira semântica baseia-se em um único contador de processos, compartilhados por todos os processos envolvidos. O contador registra o número de processos que ainda não se apresentaram na barreira. A função de defina_barreira é inicializar o contador com o valor n. Processos se apresentam na barreira invocando espera_barreira, cuja função é decrementar o contador e aguardar que ele atinja o valor zero. A implementação de espera_barreira requer regiões críticas, pois dois processos podem tentar decrementar o contador simultaneamente. A região crítica (devido ao contador único) é responsável pela complexidade O(n) desta implementação, visto que se n processos atingirem a barreira simultaneamente, o acesso à região crítica será sequencial.

Barreiras com a segunda semântica tem implementação similar, exceto que é necessário distinguir o processo privilegiado. Para tanto, o processo privilegiado é o único que aguarda que o contador seja anulado. Todos os outros processos decrementam o contador e aguardam que uma única variável lógica (barreira aberta ou fechada), compartilhada por todos, seja alterada. Quando o contador atinge o valor zero, o processo privilegiado é liberado. Uma de suas ações

futuras é invocar abra_barreira, que por sua vez alterará a variável lógica, liberando todos os processos. Observe que a introdução da variável lógica não acarreta novas regiões críticas, visto que apenas um processo altera seu conteúdo.

Uma observação a parte é que o uso repetitivo da mesma barreira exige maiores cuidados com a sua implementação. Resumidamente, deve-se evitar que processos utilizando a barreira pela segunda vez a encontrem aberta, devido a processos que ainda estão utilizando a barreira pela primeira vez. Para tanto, duplica-se a estrutura de dados, alternando-se qual delas é ativa.

3 - BARREIRAS SEQUENCIAIS: UM GARGALO DE PROCESSAMENTO PARALELO

Vejamos o impacto de barreiras sequenciais em processamento paralelo. Suponha muitos processos com poucas operações a realizar, isto é, deseja-se realizar uma computação de baixa granularidade em uma máquina paralela com muitos processadores. Neste cenário é possível que o custo da computação seja dominado pelo custo da sincronização, conforme os argumentos a seguir:

Seja $k_1 s^p$ o custo de uma computação sequencial. O custo ótimo da computação paralela "equivalente", utilizando n processadores, é $k_1 s^p/n$. Se o custo de sincronização for $k_2 n$, o custo total da computação paralela é $k_1 s^p/n+k_2 n$. Para computações de tamanho fixo (s e p fixos) o aumento de processadores acima de um valor limite implica que a sincronização dominará o custo computacional.

Um dos objetivos deste trabalho é elevar o valor limite e atenuar o custo de sincronização, substituíndo-se o fator linear por um fator logaritmico. Observe que o fator linear deve-se ao uso de regiões críticas, que são necessárias devido à unicidade do contador e à quantidade de processos que o altera. Se houvessem diversos contadores, o número de processos a atualizar cada contador seria diminuido, e diversos contadores seriam atualizados simultaneamente.

A essência deste trabalho consiste em levar esta idéia ao extremo, proporcionando um "contador" para cada processo. Como "contadores" são individuais, eles podem ser substituídos por denotadores de presença (variáveis lógicas), e a questão é como detectar a presença de processos maximizando o paralelismo, e portanto, minimizando custos. Propõe-se que os processos sejam arranjados em uma árvore binária, conforme descrição a seguir.

4 - UMA BARREIRA PARALELA

Descreve-se, a seguir, uma implementação paralela para a operação espera_barreira, utilizando-se a segunda semântica. Suponha n processos numerados de 0 a n-1, n denotadores de presença (variáveis lógicas) e um único indicador global do estado da barreira. Inicialmente, os processos são agrupado dois a dois, com o processo número 2k associado ao processo número 2k+1, para $k=1\ldots,\frac{n-1}{2}$. Ao atingir a barreira, cada processo par aguarda a chegada do processo ímpar correspondente, verificando o estado do denotador de presença associado ao processo ímpar. Processos ímpares alteram o seu denotador de presença e aguardam a abertura da barreira, monitorando o indicador global do estado da barreira.

Ao término deste passo, metade dos processos (os ímpares) estarão sincronizados, aguardando a abertura da barreira. A outra metade (os pares) estará necessitando de sincronização. Portanto, recaímos no mesmo problema, mas com metade dos processos. Utiliza-se o mesmo procedimento, com os processos de número 4k associados aos processos número 4k+2 para $k=0,\ldots,\frac{n-1}{4}$, e assim por diante.

Ao término de $\log_2 n$ passos, todos os processos estarão aguardando a liberação na barreira, exceto o processo número zero (privilegiado) que

continuará sua computação. Futuramente, este processo abrirá a barreira, invocando abra_barreira.

São duas as vantagens deste procedimento em processamento paralelo Se houver n processadores disponíveis, todos os processos ímpares podem denotar sua presença simultaneamente, pois atuam em variáveis lógicas distintas e independentes. Da mesma maneira, todos os processos pares podem aguardar a presença de seus correspondentes ímpares simultaneamente. Se todos os processos atingirem a barreira simultaneamente, o custo de cada passo é constante, e a complexidade da barreira é $O(\log_2 n)$.

A segunda vantagem é que regiões críticas são desnecessárias, visto que nenhuma das variáveis compartilhadas é alterada por mais de um processo. Cada denotador de presença é alterado por um único processo, e o estado da barreira é alterado apenas pelo processo privilegiado.

5 - RESULTADOS EXPERIMENTAIS

A barreira paralela recém-descrita foi programada em C, e mediu-se o tempo de retenção dos processos na barreira. Estes experimentos foram realizados em duas máquina MIMD de memória central: o Sequent Balance 8000, composto de 12 processadores e o Encore Multimax, composto de 20 processadores.

O tempo de liberação na barreira (em micro segundos) em função de número de processos é apresentado a seguir:

N	SEQUENT	ENCORE	(*)
1	0,50	0,55	0,87
2	0,88	0,90	1,14
3	1,16	0,98	1,46
4	1,21	1,08	1,79
5	1,47	1,27	2,06
6	1,61	1,30	2,45
7	1,69	1,31	2,75
8	1,79	1,35	3,18
9	2,05	1,50	3,44
10	2,16	1,52	3,84
11	2,27	1,56	4,07
12	2,56	1,61	4,80

Os dados comprovam o comportamento logaritmico do algoritmo paralelo. A coluna identificada por (*) consiste de tempos de execução, no Sequent, da barreira sequencial fornecida pelo fabricante. Observe que mesmo para poucos processos, a barreira sequencial demanda maior tempo de execução que a barreira paralela.

6 - CONCLUSÃO

Descreveu-se uma implementação paralela do mecanismo de barreiras, com complexidade $O(\log_2 n)$ e com tempo de execução inferior à implementação tradicional, mesmo para um pequeno número de processos. O mecanismo apresentado pode ser utilizado na implementação de outras primitivas de sincronização e do sistema operacional, atenuando o efeito do custo de sincronização em computações paralelas de baixa granularidade.