MAC0422 - Sistemas Operacionais

Daniel Macêdo Batista

IME - USP, 8 de Outubro de 2020

Roteiro

Barreiras de sincronização

Melhorando a barreira em árvore

Barreiras de sincronização

Flags e coordenadores

- Algoritmo do coordenador

```
for [i = 1 to n] while (arrive[i] == 0) skip;
for [i = 1 to n] continue[i] = 1;
```

- Há o risco de `arrive[i]` mudar depois que o coordenador passou por `i` no laço do `for`?

Flags e coordenadores

- Algoritmo do Worker

```
arrive[i] = 1;
while (continue[i] == 0) skip;
```

- Algoritmo do Coordenador

```
for [i = 1 to n] while (arrive[i] == 0) skip;
for [i = 1 to n] continue[i] = 1;
```

- Sem FA e sem contenção de memória
- Falta zerar as variáveis

Flags e coordenadores

- Primeira tentativa para zerar as variáveis

```
/* Worker */
arrive[i] = 1;
while (continue[i] == 0) skip;
arrive[i] = 0;
```

```
/* Coordenador */
for [i = 1 to n] while (arrive[i] == 0) skip;
for [i = 1 to n] continue[i] = 1;
for [i = 1 to n] continue[i] = 0;
```

- Problemas?

Flags e coordenadores

- Flag: Variável "erguida" por uma thread para sinalizar que uma condição de sincronização é verdadeira
- **Princípios da sincronização por flags**
  - A thread que espera uma flag ser 1 é a thread que zera aquela flag
  - Uma flag não pode ter valor 1 antes de se ter certeza que ela foi zerada

Flags e coordenadores


```
Thread Worker[i = 1 to n] {
   while (true) {
      codigo da tarefa i;
      arrive[i] = 1;
      while (continue[i] == 0) skip;
      continue[i] = 0;
}

Thread Coordinator {
   while (true) {
      for [i = 1 to n] {
         while (arrive[i] == 0) skip;
         arrive[i] = 0;
      }
      for [i = 1 to n] continue[i] = 1;
   }
}
```

Flags e coordenadores

- O algoritmo anterior exige uma thread a mais que só coordena ("perde" um processador)
- O tempo de execução de cada iteração do coordenador é diretamente proporcional a `n`
  - Mesmo código para os Workers
  - Provavelmente todos os Workers vão chegar na barreira na mesma hora
- O ideal seria ter a verificação feita pelo coordenador em paralelo

Flags e coordenadores


```
Thread Coordinator {
   while (true) {
      for [i = 1 to n] {
         while (arrive[i] == 0) skip;
         arrive[i] = 0;
      }
      for [i = 1 to n] continue[i] = 1;
   }
}
```

- Que parte do algoritmo pode ser paralelizada? (Onde podem ser criadas mais threads?)

Flags e coordenadores


```
Thread Coordinator {
   while (true) {
      for [i = 1 to n] {
         while (arrive[i] == 0) skip;
         arrive[i] = 0;
      }
      for [i = 1 to n] continue[i] = 1;
   }
}
```

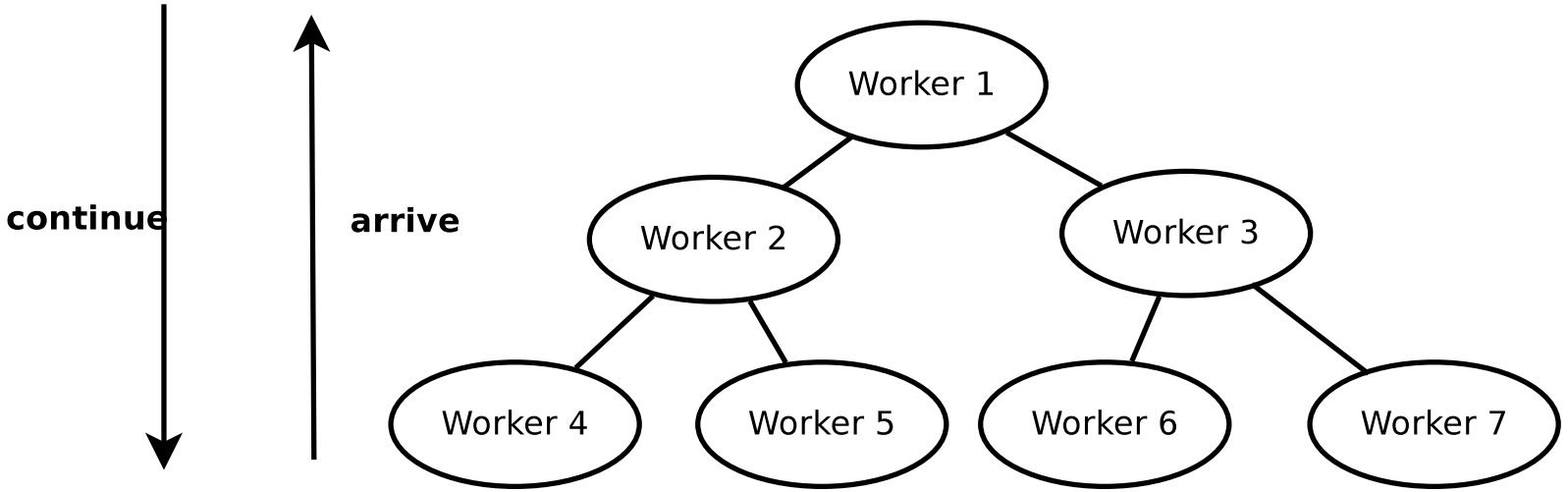
- `n` threads novas para os loops mas isso é custoso :(
- Mas já temos `n` threads!

Flags e coordenadores

- Ideia: usar os workers como coordenadores também
- Organizá-los em alguma estrutura que permita a comparação em paralelo ao invés de sequencial

Flags e coordenadores

- Barreira em árvore



- O algoritmo tem que ser diferente para folha, nó interno e raiz

Flags e coordenadores


```
folha L: arrive[L] = 1;
         while (continue[L] == 0) skip;
         continue[L] = 0;

no interno I: while (arrive[left] == 0) skip;
              arrive[left] = 0;
              while (arrive[right] == 0) skip;
              arrive[right] = 0;
              arrive[I] = 1;
              while (continue[I] == 0) skip;
              continue[I] = 0;
              continue[left] = 1; continue[right] = 1;

no raiz R: while (arrive[left] == 0) skip;
           arrive[left] = 0;
           while (arrive[right] == 0) skip;
           arrive[right] = 0;
           continue[left] = 1; continue[right] = 1;
```

Flags e coordenadores

- No exemplo anterior, todos estão esperando o `continue` começar a ser modificado pela raiz e ele vai sendo propagado aos poucos pela árvore
- O código pode ser melhorado se o processo raiz enviar um `continue` em broadcast

Melhorando a barreira em árvore

Com o `continue` em broadcast


```
folha L: if (round == 0) {
           arrive[L] = 1;
           while (continue == 0) skip;
           round = 1;
         }
         else {
           arrive[L] = 1;
           while (continue == 1) skip;
           round = 0;
         }

no interno I: ...

no raiz R: while (arrive[left] == 0) skip;
           arrive[left] = 0;
           while (arrive[right] == 0) skip;
           arrive[right] = 0;
           if (round == 0) { round = 1; continue = 1; }
           else { round = 0; continue = 0;}
```

- round é local!