PROGRAMAÇÃO DINÂMICA EM ARQUITETURAS PARALELAS: ANÁLISE DA COMPLEXIDADE

Laira Vieira Toscani
Univ. Fed. do Rio Grande do Sul
CX. Postal 1501, 90001 - Porto Alegre - RS

Paulo Augusto S. Veloso
Pont. Univ. Cat. do Rio de Janeiro
Rua Marques de São Vicente, 225 - Deptº de Inf.
22453 - Rio de Janeiro, RJ

SUMÁRIO

Este trabalho parte de uma formalização da programação dinâmica e examina duas arquiteturas paralelas com o objetivo de melhorar a eficiência do método. É apresentada uma análise de complexidade das versões seqüencial e paralela, juntamente com alguns exemplos ilustrativos.

ABSTRACT

In this work we start from a formal specification of the dynamic programming method and examine two parallel architectures to improve the performance of the method. We present a complexity analysis of the sequential and parallel versions of the dynamic programming method. Some illustrative examples are presented.

1. INTRODUÇÃO

A primeira proposta de um algoritmo paralelo para programação dinâmica é a de J. Casti [1], onde foi adotado o modelo ILLIAC IV. Bertolazzi [2] apresentou um esboço de algoritmo paralelo para a programação dinâmica, para arquiteturas Mestre/Escravo e Array Circular, com a análise da complexidade resultante em cada modelo, calculada em função do número de estados admissíveis, decisões admissíveis e número de estágios do algoritmo.

Este trabalho usa os mesmos modelos de arquitetura usados por Bertolazzi e a formalização da programação dinâmica via tipos abstratos de dados apresentada em [3], para expressar a complexidade dos algoritmos em termos da complexidades de funções primitivas (também definidas em [3]), determinar as condições que tornam um modelo melhor que outro e definir as condições de máximo aproveitamento dos processadores disponíveis. Os resultados são tabulados e ilustrados com exemplos clássicos de programação dinâmica.

A notação para especificar a comunicação entre dois processos, utilizada nos programas paralelos, foi criada por Hoare [4]. Sua sintaxe é descrita pela seguinte BNF:

<comando-de-entrada>: := <fonte>?<lista-de-variáveis>

<comando-de-saída> : := <destino>!<lista-de-expressões>

<fonte> : := <nome-do-processador>

<destino> : := <nome-do-processador>

A comunicação entre dois processadores se efetua quando: (a) o comando de entrada para o primeiro processador especifica como fonte o nome do segundo processador, (b) o comando de saída do segundo processador especifica como destino o nome do primeiro processador, e (c) a lista de variáveis do comando de entrada é coerente com a lista de expressões do comando de saída.

2. PROGRAMAÇÃO DINÂMICA EM ARQUITETURA TIPO MESTRE/ESCRAVO

O Modelo Mestre/Escravo consiste de um processador Mestre e p processadores Escravos. Toda comunicação entre processos é feita através do processador Mestre. Este modelo é vantajoso quando o tempo de computação das funções em paralelo é grande e a necessidade de comunicação é pequena.

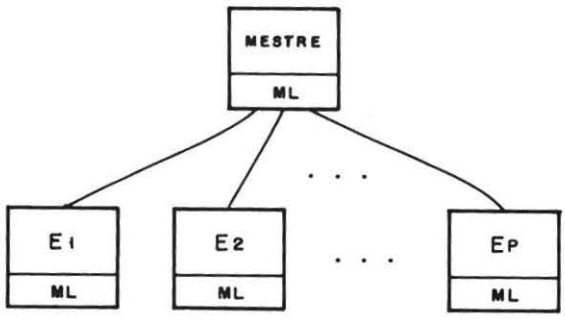

Fig. 1: Modelo Mestre/Escravo

O processador Mestre executará um programa e os processadores Escravos executarão todos o mesmo programa. Os programas seguintes são versões adaptadas à arquitetura do programa abstrato apresentado em [3].

Para o processador Mestre:

```
1.  entrada: d
2.  n ← tamanho(d)
3.  q ← decompõe(d)
4.  mínimo ← tamanho+(q)
5.  m ← inicializa(q)
6.  q ← combina(q,m)
7.  para k=mínimo até n-1 faça
8.        nk ← número de elementos
                da seqüência q
9.        para j=0 até ⌊nk/p⌋ faça
10.             para i = j*p+1 até
                    min {(j+1)*p, nk} faça
11.                 P(i-j*p) ! (qi, m)
12.             fim - para
13.       fim - para
14.       para j=0 até ⌊nk/p⌋ faça
15.             para i=j*p+1 até
                    min {(j+1)*p, nk faça
16.                  P(i-j*p) ? mi'
17.             fim - para
18.       fim - para
19.       q' ← combina (q, m)
20.       (q, m) ← (q', m')
21. fim - para
22. r ← recupera (m)
23. saída: r
```

Para o processador Pi

```
1.  Mestre ?(qi, m)
2.  mi' ← altera (qi, m)
3.  Mestre ! mi'
```

O processador Mestre recebe o problema a ser resolvido, o decompõe em uma seqüência q de problemas de tamanho mínimo, resolve esses problemas (inicializa). Então, inicia uma iteração que a cada passo envia p problemas (i=j*p+1 até (j+1)*p) um para cada processador, até se esgotar a seqüência q; espera as respostas, combina os resultados e prepara a nova seqüência de problemas a serem resolvidos. Cada processador escravo resolve o problema (função altera) e envia a resposta para o Mestre. No final da iteração o processador Mestre recupera o resultado que deve estar na variável m.

2.1. Complexidade

Vamos usar em todo trabalho Σ para simbolizar $\sum_{k=mínimo}^{n-1}$, c(f) para denotar complexidade da função f e c(básicas) para representar c(tamanho)+ +c(decompõe)+c(tamanho$^+$)+c(recupera). CompME a complexidade do programa é dada pela expressão:

$$\text{CompME}(n) = \underbrace{c(\text{básicas})(n)}_{1:1\text{-}4;22\text{-}23} + \underbrace{c(\text{inicializa})(n)}_{1\ \ 5} +$$

$$+ \Sigma \{ \underbrace{\frac{nk}{p}\, p}_{1:9\text{-}13} + \underbrace{\frac{nk}{p} \cdot p}_{1:14\text{-}18}$$

$$+ \underbrace{(\frac{nk}{p} c(\text{altera})(nk))\}}_{1\ \ 16}$$

$$= c(\text{básicas})(n) + c(\text{inicializa})(n) +$$

$$+ \frac{1}{p} \Sigma\, nk\ c(\text{altera})(nk) + \Sigma\, nk$$

As linhas 6,19 e 20 tem o objetivo de facilitar o entendimento do programa. A função combina tem somente o efeito de formalizar a criação de uma nova seqüência de subproblemas a serem resolvidos; não é executada e portanto não é considerada para efeito de cálculo de complexidade. Somente uma variável m é necessária, pois as posições de m alteradas em uma iteração não são acessadas na mesma iteração. Assim a linha 20 também não é considerada no cálculo da complexidade.

2.1. Exemplos

Aqui serão apresentados cinco exemplos: os três primeiros PCV, PABBO e POOMM estão formalizados e analisados em [3], os outros dois PCMCEV e MOCHILA em [5].

Problema do Caixeiro Viajante (PCV)

A programação dinâmica é especialmente útil no projeto de algoritmos para problemas NP-difíceis, como o PCV, por exemplo, cujo algoritmo trivial é $O(n!)$ e o algoritmo obtido através dessa técnica é $O(n^2 2^n)$ conforme mostrado em [3]. Este mesmo algoritmo executado numa máquina Mestre/Escravo pode diminuir a ordem de complexidade, como será visto a seguir.

$c(\text{inicializa})(n) = O(n)$, $c(\text{básicas})(n) = O(n)$, mínimo = 2, $c(\text{altera})(k) = k-1$ e $nk = (n-1)\binom{n-2}{k-1}$

$$\text{CompME} = O(n) + \frac{1}{p} \sum_{k=2}^{n-1} \binom{n-2}{k-1} (n-1)(k-1)$$

$$+ \sum_{k=2}^{n-1} (n-1)\binom{n-2}{k-1}$$

$$= O(n) + \frac{1}{p}\ O(n^2 2^n) + (n-1)2^{n-2}$$

$$= \frac{1}{p}\, O(n^2 2^n) + O(n2^n)$$

Independentemente do número de processadores, CompME não baixará de $O(n2^n)$ e $\text{CompME} = O(n2^n)$ para $p = O(n)$.

A complexidade do mesmo algoritmo no modelo seqüencial CompSEQ é $O(n^2 2^n)$.

Assim: $\text{CompME} < \text{CompSEQ}$ se $\lim_{n\to\infty} p = \infty$ (onde "<" significa menor em ordem de complexidade).

Problema da Árvore de Busca Binária Ótima (PABBO)

Existem $4^n/n\sqrt{\pi n}$ árvores binárias distintas com n elementos [6], logo um algoritmo que enumerasse todas as árvores e as comparasse seria de ordem exponencial e portanto inviável para n relativamente grande. Com programação dinâmica che-

ga-se a um algoritmo polinomial e o paralelismo ainda pode diminuir a ordem de complexidade.

c(inicializa)(n) = 0(n), c(básicas)(n) = 0(1), mínimo = 0, c(altera)(k) = 0(k) e nk = n-k+1

$$\text{CompME} = 0(n) + \frac{1}{p}\sum_{k=0}^{n-1}(n-k+1)k + \sum_{k=0}^{n-1} n-k+1$$

$$= 0(n) + \frac{1}{p}\ 0(n^3) + 0(n^2)$$

$$= \frac{1}{p}\ 0(n^3) + 0(n^2)$$

Como no exemplo anterior, com p = 0(n) é atingido o ganho máximo, com CompME = $0(n^2)$. CompSEQ = $0(n^3)$ e CompME < CompSEQ se $\lim_{n\to\infty} p=\infty$

Problema da Ordem Ótima da Multiplicação de Matrizes (POOMM)

Um algoritmo que enumere todas as seqüências possíveis de multiplicações, calcule os respectivos números totais de operações requeridos e, em seguida, escolha uma seqüência ótima terá complexidade exponencial em n (número de matrizes). Utilizando a programação dinâmica pode-se diminuir a ordem de complexidade para uma ordem polinomial. Com o uso de paralelismo a ordem de complexidade pode ainda diminuir.

c(inicializa)(n) = 0(n), c(básicas)(n) = 0(1), mínimo = 0, c(altera)(k)= 0(k) e n_k = n-k

$$\text{CompME} = 0(n) + \frac{1}{p}\sum_{k=0}^{n-1}(n-k)k + \sum_{k=0}^{n-1}(n-k)$$

$$= 0(n) + \frac{1}{p}\ 0(n^3) + 0(n^2)$$

$$= \frac{1}{p}\ 0(n^3) + 0(n^2)$$

Novamente, independente do número de processadores CompME não baixa de $0(n^2)$ e CompME=$0(n^2)$ para p = 0(n).

Além disso, CompSEQ = $0(n^3)$ e CompME < CompSEQ se $\lim_{n\to\infty} p = \infty$

Problema do Caminho Mais Curto entre Vértices (PCMCEV)

O algoritmo trivial é exponencial e com programação dinâmica consegue-se um algoritmo $0(n^3)$. Com paralelismo esta marca pode baixar.

c(inicializa)(n) = $0(n^2)$, c(básicas)(n) = 0(1), mínimo = 0, c(altera)(k,n) = 0(n), nk = n

$$\text{CompME} = 0(n^2) + \frac{1}{p}\sum_{n=0}^{n-1} n\ 0(n) + \sum_{k=0}^{n-1} n$$

$$= 0(n^2) + \frac{1}{p}\ 0(n^3) + 0(n^2)$$

$$= \frac{1}{p}\ 0(n^3) + 0(n^2)$$

Se p = 0(n) CompME = $0(n^2)$, este é o melhor desempenho do algoritmo no modelo Mestre/Escravo. CompSEQ = $0(n^3)$ e CompME < CompSEQ se $\lim_{n\to\infty} p = \infty$

Problema da Mochila (MOCHILA)

Em alguns casos, como neste, o paralelismo resultante da arquitetura Mestre/Escravo proposta não diminui a ordem de complexidade do algoritmo.

c(inicializa)(n) = $0(n^2)$, c(básicas)(n) = 0(1), mínimo = 0, c(altera)(k) = 0(1) e nk = 2^{n-k}

$$\text{CompME} = 0(2^n) + \frac{1}{p}\sum_{k=0}^{n-1} 2^{n-k} + \sum_{k=0}^{n-1} 2^{n-k}$$

$$= 0(2^n) + \frac{1}{p}\ 0(2^n) + 0(2^n)$$

$$= \frac{1}{p}\ 0(2^n) + 0(2^n)$$

Para $p \leq 0(2^n)$, CompME = $0(2^n)$.

CompSEQ = $0(2^n)$ e a condição CompME < CompSEQ nunca é atingida. O ganho na resolução de 2^n problemas em paralelo, que não é muito porque c(altera) = 0(1), é compensado na comunicação dos resultados ao processador Mestre e na atualização por este da memória de cada processador Escravo.

O modelo Mestre/Escravo não é adequado, portanto, a este problema. Na próxima seção será apresentado um outro modelo de arquitetura, mais adequado para o problema MOCHILA.

3. PROGRAMAÇÃO DINÂMICA EM ARQUITETURA ARRAY CIRCULAR

Suponha o seguinte modelo: uma máquina SIMD na qual cada processador é controlado por uma unidade de controle, que distribui a mesma seqüência de instruções para todos processadores. Cada processador tem uma memória local e está ligado com seus dois vizinhos numa arquitetura tipo Array Circular.

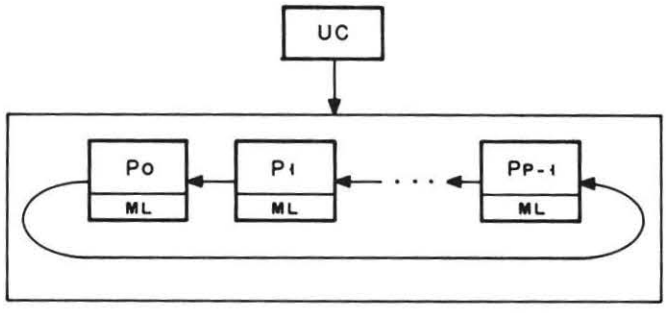

Fig. 2: Modelo Array Circular

Cada processador executará o mesmo programa:

```
1. entrada: d
2. n ← tamanho(d)
3. q ← decompõe(d)
4. mínimo ← tamanho+(q)
5. n' ← comprimento de seqüência q
/São inicializados ⌈n'/p⌉ problemas (inicializa
(q) = (inic(q1),...inic(qn'))/
6. para k' = j+1 até ⌊n'/p⌋ * p+j+1 passo: p
       faça
7.       se k' ≤ n' então m'k'←inic(qk')
8.                 senão m'k'←Λ
9. fim - para
/ ⌈n'/p⌉ valores são enviados para o processa-
   dor da esquerda, o mesmo número de va-
   lores são recebidos do processador da
   direita; este processo se repete p-1
   vêzes/
10. para i=0 até p-2 faça
11.   para k' =(j+1)modp +1 até
          (j+i)modp +1 + ⌊n'/p⌋ * p passo: p faça
12.       P(j-1) modp ! m'k'
13.       fim - para
14.       para k'=(j+i+1) modp +1 até
              (j+i+1)modp +1 + ⌊n'/p⌋ * p passo:
              p faça
15.           P(j+1) mod p ? m'k'
16.       fim - para
17. fim - para
18. q ← combina (q, m)
19. para k = mínimo até n-1 faça
/   Se repetem as mesmas linhas de programa, da
    linha 5 a linha 17 trocando na linha 7 a
    função inic pela função altera/
33.       q' ← combina (q, m)
34.       (q, m) ← (q', m')
35. fim - para
36. r ← recupera (m)
37. saída: r
```

Cada processador executa o mesmo programa: decompõe o problema original numa seqüência q de n' problemas, resolve (função inicializa) $\lceil n'/p \rceil$ problemas, manda essas soluções para o processador da esquerda, recebe $\lceil n'/p \rceil$ soluções do processador da direita, envia-os também para o processador da esquerda até que os p processadores tenham m inicializada, inicia então uma iteração (k = mínimo até n-1) em que a cada passo são resolvidos (função altera) $\lceil n'/p \rceil$ problemas e as soluções transmitidas, como na fase de inicialização; ao final da iteração todos processadores tem m atualizado e podem recuperar a solução do problema original.

3.1. Complexidade

A complexidade total neste modelo é

$$\text{CompAC}(n) = \underbrace{c(\text{básicas})(n)}_{1:1\text{-}5;36} + \frac{1}{p}\underbrace{c(\text{inicializa})(n)}_{1:6\text{-}17} + \underbrace{O(p)}_{} + \underbrace{\frac{1}{p}\sum nk\ c(\text{altera})(k,n) + \sum O(p)}_{1:19\ -\ 35}$$

3.2. Exemplos

Para o PCV:

$$\text{CompAC}(n) = O(n) + \frac{1}{p} O(n) + \sum_{k=2}^{n-1} (n-1)\ \binom{n-2}{k-1}(k-1) + n\ O(p)$$

$$= n\ O(p) + \frac{1}{p}\ O(n^2\ 2^n)$$

Para o PABBO:

$$\text{CompAC}(n) = \frac{1}{p}\ O(n) + \frac{1}{p} \sum_{k=0}^{n-1} (n-k+1)\ O(1) + nO(p)$$

$$= nO(p) + \frac{1}{p}\ O(n^3)$$

Para o POOM:

$$\text{CompAC}(n) = \frac{1}{p}\ O(n) + \frac{1}{p} \sum_{k=0}^{n-1} (n-k)\ O(k) + n\ O(p)$$

$$= nO(p) + \frac{1}{p}\ O(n^3)$$

Para o PCMCEV:

$$\text{CompAC}(n) = \frac{1}{p}\ O(n^2) + \frac{1}{p} \sum_{k=0}^{n-1} O(n^2) + nO(p)$$

$$= n\ O(p) + \frac{1}{p}\ O(n^3)$$

Para a MOCHILA:

$$\text{CompAC} = O(1) + \frac{1}{p}\ O(2^n) + O(p) + \frac{1}{p} \sum_{k=0}^{n-1} 2^{n-k} + \sum_{k=0}^{n-1} (p)$$

$$= nO(p) + \frac{1}{p}\ O(2^n)$$

$\text{CompAC} < \text{CompSEQ}$ se $p < O\left(\frac{2^n}{n}\right)$ e $\lim_{n\to\infty} p = \infty$

4. CONCLUSÕES

O Modelo Mestre/Escravo mostrou-se adequado para programação dinâmica, exceto quando nk (número de subproblemas) é bem maior que a complexidade para resolver um desses problemas (neste caso o tempo de comunicação supera o tempo ganho resultante do paralelismo).

Neste trabalho foram estudados cinco exemplos. Para os quatro primeiros (PCV, PABBO, POOMM, PCMCEM) n processadores são suficientes para atingir a complexidade ótima no modelo Mestre/Escravo e o ganho é também ótimo

$$(\text{CompME} = \frac{1}{p}\ \text{CompSEQ}).$$

Para o problema MOCHILA o modelo Mestre/Escravo não é conveniente, pois não permite ganho em ordem de complexidade. No modelo Array Circular o algoritmo para o problema MOCHILA atingiu ganho ótimo com p = n.

Em resumo, a programação dinâmica foi analisada quanto a sua complexidade em três modelos de arquitetura: seqüencial (CompSEQ), Mestre/Escravo (CompME) e Array Circular (CompAC). Os resultados são sintetizados em quatro tabelas.

Os cinco exemplos podem ser generalizados para quatro casos cujas complexidades, nos três modelos, são apresentadas nas tabelas 1 e 2.

Na tabela 3, para os cinco exemplos, são apresentadas as complexidades CompSEQ, CompME para p=n e CompAC para p=n. Também são apresentados os ganhos dos dois modelos paralelos com p=n, em relação ao modelo seqüencial, e o desempenho ótimo de cada exemplo nos modelos paralelos.

Na tabela 4 são comparadas as complexidades nos três modelos. Foram consideradas cinco situações interessantes e apresentadas as condições para serem atingidas tais situações. Para o modelo Array Circular a condição de ganho ótimo $(\text{CompAC} = \frac{1}{p}\ \text{CompSEQ})$ é simples $(p^2 \leq \frac{1}{n}\text{CompSEQ})$; assim, aumentando o número de processadores é possível diminuir a complexidade de execução até o limite de $O(2\sqrt{n\ \text{CompSEQ}})$ com $p^2 = \frac{1}{n}\ \text{CompSEQ}$.

As deduções dos resultados aqui apresentados estão descritas detalhadamente em [5].

REFERÊNCIAS BIBLIOGRÁFICAS

[1] CASTI, J.; RICHARDSON, M. & LARSON, R. Dynamic Programming and Parallel Computers, Journal of Optimization Theory and Applications. Vol. 12, n.4, 1973.

[2] BERTOLAZZI, P. & PIROZZI, M., Parallel Algorithms for Dynamic Programming Problems. Elaboratori Paralleli e Calcolo Scientifica, Roma, 1982.

[3] TOSCANI, Laira V. & VELOSO, Paulo A.S., Especificação Formal e Análise da Complexidade da Programação Dinâmica. Porto Alegre, CPGCC da UFRGS, 1986. (RP nº 49)

[4] HOARE, C.A.R., "Communicationg Sequential Processes", Communication of the ACM 21, pp 666-77, 1978.

[5] TOSCANI, L.V. & VELOSO, P.A.S., A Programação Dinâmica em Arquiteturas Paralelas. (a ser publicado)

[6] TERADA, R., Desenvolvimento de Algoritmos e Complexidade de Computação. Departamento de Informática, PUC/RJ, 1982.

Caso	c(básicas) + c(inicializa)	c(altera)	nk	CompSEQ	CompME	CompME < CompSEQ
1	$O(n)$	$O(1)$	$O(n)$	$O(n^2)$	$\frac{1}{p}\ O(n^2) + O(n^2)$	Impossível
2	$O(n)$	$O(k^r)$ $r\geq 1$	$n-k$	$O(n^{r+2})$	$\frac{1}{p}\ O(n^{r+2}) + O(n^2)$	$\lim_{n\to\infty} p = \infty$
3	$O(n)$	$O(k)$	$\binom{n}{k}$	$O(n2^n)$	$\frac{1}{p}\ O(n2^n) + O(2^n)$	$\lim_{n\to\infty} p = \infty$
4	$O(n)$	$O(k)$	2^{n-k}	$O(2^n)$	$\frac{1}{p}\ O(2^n) + O(2^n)$	Impossível
PCV	$O(n)$	$O(k)$	$\binom{n-2}{k-1}(n-1)$	$O(n^2 2^n)$	$\frac{1}{p}\ O(n^2 2^n) + O(n2^n)$	$\lim_{n\to\infty} p = \infty$
PABBO	$O(n)$	$O(k)$	$n-k+1$	$O(n^3)$	$\frac{1}{p}\ O(n^3) + O(n^2)$	$\lim_{n\to\infty} p = \infty$
POOMM	$O(n)$	$O(k)$	$n-k$	$O(n^3)$	$\frac{1}{p}\ O(n^3) + O(n^2)$	$\lim_{n\to\infty} p = \infty$
PCMCEV	$O(n^2)$	$O(n)$	n	$O(n^3)$	$\frac{1}{p}\ O(n^3) + O(n^2)$	$\lim_{n\to\infty} p = \infty$
MOCHILA	$O(2^n)$	$O(1)$	2^{n-k}	$O(2^n)$	$\frac{1}{p}\ O(2^n) + O(2^n)$	Impossível

Tabela 1: CompSEQ e CompME

* Comparação em Ordem de Complexidade

Caso	c(básicos)	c(inicializa)	c(altera)	nk	CompAC	CompAC < CompME	CompAC < CompSEQ
1	$O(n)$	$O(n)$	$O(1)$	$O(n)$	$\frac{1}{p} O(n^2) + n\,O(p)$	$p < O(n)$ e $\lim_{n\to\infty} p = \infty$	$p < O(n)$ e $\lim_{n\to\infty} p = \infty$
2	$O(n)$	$O(n)$	$O(k^r)$	$n-k$	$\frac{1}{p} O(n^{r+2}) + n\,O(p)$	$p < O(n)$, $r = 0$ e $\lim_{n\to\infty} p = \infty$	$p < O(n^{r+1})$ e $\lim_{n\to\infty} p = \infty$
3	$O(n)$	$O(n)$	$O(k)$	$\binom{n}{k}$	$\frac{1}{p} O(n2^n) + n\,O(p)$	$p > O(n)$ e $p < O(\frac{2^n}{n})$	$p < O(2^n)$ e $\lim_{n\to\infty} p = \infty$
4	$O(1)$	$O(2^n)$	$O(k)$	2^{n-k}	$\frac{1}{p} O(2^n) + n\,O(p)$	$p < O(\frac{2^n}{n})$ e $\lim_{n\to\infty} p = \infty$	$p < O(\frac{2^n}{n})$ e $\lim_{n\to\infty} p = \infty$
PCV	$O(n)$	$O(n)$	$k-1$	$(n-1)\binom{n-2}{k-1}$	$\frac{1}{p} O(n^2 2^n) + n\,O(p)$	$p < O(2^n)$ e $p > O(n)$	$p < O(n\,2^n)$ e $\lim_{n\to\infty} p = \infty$
PABBO	$O(1)$	$O(n)$	$O(k)$	$n-k+1$	$\frac{1}{p} O(n^3) + n\,O(p)$	impossível	$p < O(n^2)$ e $\lim_{n\to\infty} p = \infty$
POOMM	$O(1)$	$O(n)$	$O(k)$	$n-k$	$\frac{1}{p} O(n^3) + n\,O(p)$	impossível	$p < O(n^2)$ e $\lim_{n\to\infty} p = \infty$
PCMCEV	$O(1)$	$O(n^2)$	$O(n)$	n	$\frac{1}{p} O(n^3) + n\,O(p)$	$p < O(n)$ e $\lim_{n\to\infty} p = \infty$	$p < O(n^2)$ e $\lim_{n\to\infty} p = \infty$
MOCHILA	$O(1)$	$O(2^n)$	$O(1)$	2^{n-k}	$\frac{1}{p} O(2^n) + n\,O(p)$	$p < O(2^n)$ e $\lim_{n\to\infty} p = \infty$	$p < O(2^n)$ e $\lim_{n\to\infty} p = \infty$

Tabela 2: CompAC , CompME e CompSEQ nos quatro casos e cinco exemplos

EXEMPLOS	CompSEQ	CompME(p=n)	CompAC(p=n)	OBSERVAÇÕES (p=n)	MDSME	MDSAC
PCV	$O(n^2 2^n)$	$O(n2^n)$	$O(n2^n)$	CompME = $\frac{1}{p}$ CompSEQ CompAC = $\frac{1}{p}$ CompSEQ	$O(n2^n)$	$O(n\sqrt{n}\,2^{n/2})$ com $p = O(2^{n/2}\sqrt{n})$
PABBO	$O(n^3)$	$O(n^2)$	$O(n^2)$	CompME = $\frac{1}{p}$ CompSEQ CompAC = $\frac{1}{p}$ CompSEQ	$O(n^2)$	$O(n^2)$
POOMM	$O(n^3)$	$O(n^2)$	$O(n^2)$	CompME = $\frac{1}{p}$ CompSEQ CompAC = $\frac{1}{p}$ CompSEQ	$O(n^2)$	$O(n^2)$
PCMCEV	$O(n^3)$	$O(n^2)$	$O(n^2)$	CompME = $\frac{1}{p}$ CompSEQ CompME = $\frac{1}{p}$ CompSEQ	$O(n^2)$	$O(n^2)$
MOCHILA	$O(2^n)$	$O(2^n)$	$O(\frac{2^n}{n})$	CompME = CompSEQ CompAC = $\frac{1}{p}$ CompSEQ	$O(2^n)$	$O(\sqrt{n}\,2^{n/2})$ com $p = O(\frac{2^{n/2}}{\sqrt{n}})$

Tabela 3: MDSME e MDSAC para os cinco exemplos

MDSME - Melhor desempenho no sistema M/E
MDSAC - Melhor desempenho no sistema Array Circular

SITUAÇÕES INTERESSANTES	CONDIÇÕES SUFICIENTES PARA OCORRER A SITUAÇÃO
CompME = $\frac{1}{p}$ CompSEQ	CompSEQ = c(inicializa) e $p = O(1)$ ou CompSEQ = Σ nk c(altera), c(inicializa) < $\frac{1}{p}$ Σ nk c(altera), e Σ nk < $\frac{1}{p}$ Σ nk c(altera)
CompME < CompSEQ	CompSEQ = Σ nk c(altera) > c(inicializa), c(altera) > $O(1)$ e $\lim_{n\to\infty} p = \infty$
CompAC = $\frac{1}{p}$ CompSEQ	$p^2 \leq \frac{CompSEQ}{n}$
CompAC < CompSEQ	$p^2 < \frac{CompSEQ}{n}$ e $\lim_{n\to\infty} p = \infty$
CompAC < CompME	$p < \frac{\Sigma nk}{n}$ e $\lim_{n\to\infty} p = \infty$

Tabela 4: Condições suficientes para ocorrer as situações interessantes