

## Fontes principais

1. E. Cáceres, H. Mongeli, S. Song: Algoritmos paralelos usando CGM/PVM/MPI: uma introdução  
<http://www.ime.usp.br/~song/papers/jai01.pdf>
2. E. Cáceres, C. Nishibe: 0-1 Knapsack problem: BSP/CGM Algorithm and implementation. Proc of the 17th LASTED International Conference on Parallel and Distributed Computing and System (PDCS 2005), pp. 331-335, 2005

## Programação dinâmica

## Problema da mochila 0-1

O problema da mochila 0-1 consiste de um conjunto  $S = \{1, 2, \dots, n\}$  de  $n$  itens distintos, cujo  $i$ —ésimo item possui um valor  $v_i$  e um peso  $w_i$  onde  $v_i$  e  $w_i$  são inteiros.

Seja  $W$  um inteiro que representa a capacidade máxima da mochila que será utilizada para transportar itens.

## Problema da mochila 0-1

O objetivo consiste em determinar quais itens devem ser escolhidos a fim de encher a mochila com valores mais valioso e que não exceda a capacidade máxima da mochila.

Ou seja:

$$\max\left\{\sum_{i=1}^n v_i z_i : \sum_{i=1}^n w_i z_i \leq W, z_i \in \{0, 1\}\right\}$$

## Algoritmo de Gilmore e Gomory

Primeiro algoritmo a usar programação dinâmica para resolver o problema da mochila 0-1.

Seja  $f(r, c)$ , com  $1 \leq r \leq n$  e  $0 \leq c \leq W$ , os valores da solução ótima do problema da mochila 0-1 com um conjunto de objetos  $[1, r]$  e peso  $c$ . Consequentemente,  $f(n, W)$  é o valor da solução ótima. A relação de recorrência é:

$$f(r, c) = \max\{f(r - 1, c), f(r - 1, c - w_r) + v_r\}$$

$\forall c$ , com  $0 \leq c \leq W$ , onde  $r = 1, 2, \dots, n$

Tempo:  $O(nW)$

## Problema da mochila 0-1

### Algoritmo

**para**  $c = 0$  **até**  $W$  **faça**

$f(0, c) = 0$

**para**  $r = 1$  **até**  $n$  **faça**

$f(r, 0) = 0$

**para**  $r = 1$  **até**  $n$  **faça**

**para**  $c = 1$  **até**  $W$  **faça**

**se**  $c < w_r$  **então**

$f(r, c) := f(r - 1, c)$

**senão**

$f(r, c) := \max\{f(r - 1, c - w_r) + v_r, f(r - 1, c)\}$

## Problema da mochila 0-1

Exemplo: Sejam uma mochila de capacidade 10 e 3 objetos com seus pesos e valores representados na tabela abaixo:

Objeto	1	2	3
Peso	2	3	6
Valor	3	6	9

## Problema da mochila 0-1

	0	1	2	3	4	5	6	7	8	9	10
0											
2											
3											
6											

Objeto	1	2	3
Peso	2	3	6
Valor	3	6	9



## Problema da mochila 0-1

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
2	0										
3	0										
6	0										

Objeto	1	2	3
Peso	2	3	6
Valor	3	6	9

## Problema da mochila 0-1

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
2	0	0									
3	0										
6	0										

Objeto	1	2	3
Peso	2	3	6
Valor	3	6	9

## Problema da mochila 0-1

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	3								
3	0										
6	0										

Objeto	1	2	3
Peso	2	3	6
Valor	3	6	9

## Problema da mochila 0-1

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	3	3							
3	0										
6	0										

Objeto	1	2	3
Peso	2	3	6
Valor	3	6	9

## Problema da mochila 0-1

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	3	3	3						
3	0										
6	0										

Objeto	1	2	3
Peso	2	3	6
Valor	3	6	9

## Problema da mochila 0-1

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	3	3	3	3	3	3	3	3	3
3	0										
6	0										

Objeto	1	2	3
Peso	2	3	6
Valor	3	6	9


## Problema da mochila 0-1

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	3	3	3	3	3	3	3	3	3
3	0	0	3	↓							
6	0										

Objeto	1	2	3
Peso	2	3	6
Valor	3	6	9

## Problema da mochila 0-1

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	3	3	3	3	3	3	3	3	3
3	0	0	3	6							
6	0										



Objeto	1	2	3
Peso	2	3	6
Valor	3	6	9



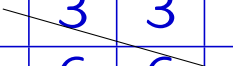
## Problema da mochila 0-1

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	3	3	3	3	3	3	3	3	3
3	0	0	3	6	6						
6	0										

Objeto	1	2	3
Peso	2	3	6
Valor	3	6	9

## Problema da mochila 0-1

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	3	3	3	3	3	3	3	3	3
3	0	0	3	6	6	9					
6	0										



Objeto	1	2	3
Peso	2	3	6
Valor	3	6	9

## Problema da mochila 0-1

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	3	3	3	3	3	3	3	3	3
3	0	0	3	6	6	9	9				
6	0										

Objeto	1	2	3
Peso	2	3	6
Valor	3	6	9

## Problema da mochila 0-1

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	3	3	3	3	3	3	3	3	3
3	0	0	3	6	6	9	9	9	9	9	9
6	0										

Objeto	1	2	3
Peso	2	3	6
Valor	3	6	9

## Problema da mochila 0-1

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	3	3	3	3	3	3	3	3	3
3	0	0	3	6	6	9	9	9	9	9	9
6	0	0	3	6	6	9	9	9	12	15	15

Objeto	1	2	3
Peso	2	3	6
Valor	3	6	9

○ algoritmo *Wavefront*

## O algoritmo *Wavefront*

Algoritmo BSP/CGM para resolver o problema da mochila, considerando  $n$  itens, capacidade  $W$  e  $p$  processadores.

- $O(p)$  rodadas de comunicação
- $O(nW/p)$  de computação

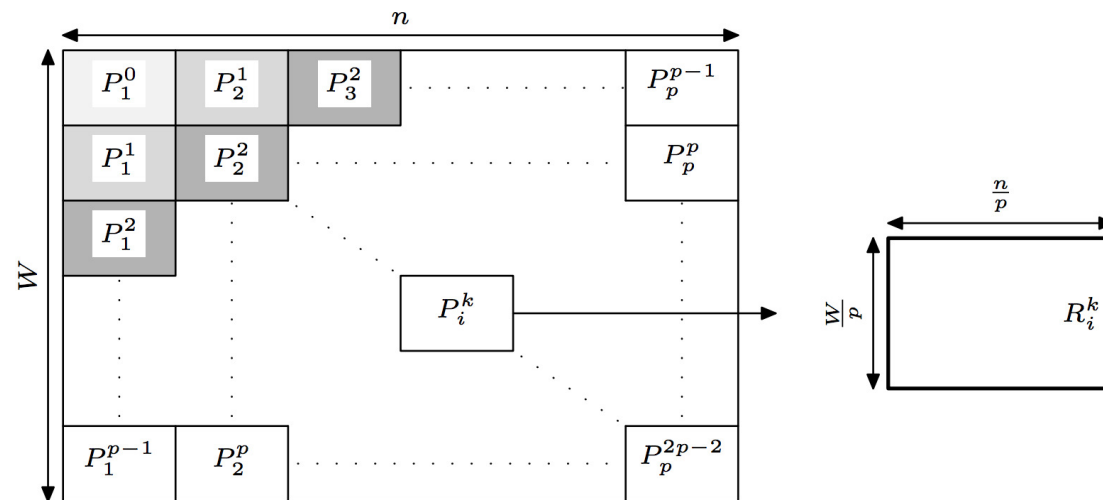
## O algoritmo *Wavefront*

Para cada conjunto  $S = \{1, 2, \dots, n\}$  de itens:

- O vetor  $w$ , onde  $w[i]$  é o peso de cada item  $i$ , é distribuído para todos os processadores.
- O vetor  $v$ , onde  $v[i]$  é o valor do item  $i$ , é dividido em  $p$  partes de tamanho  $\frac{n}{p}$ , cada processador  $P_i$ ,  $1 \leq i \leq p$ , recebe a  $i$ -ésima parte de  $v$  ( $v[(i-1)\frac{n}{p} + 1 \dots i\frac{n}{p}]$ ).

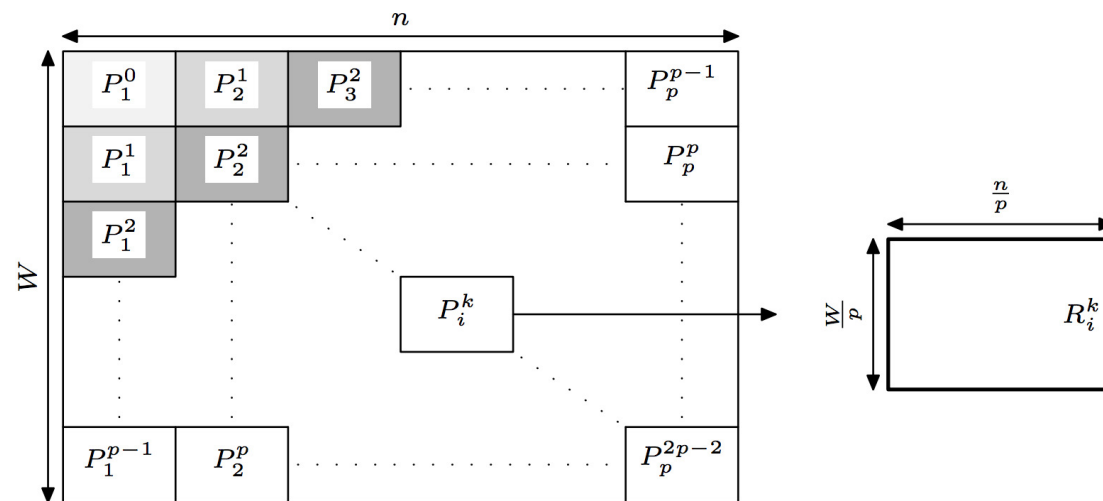


## O algoritmo *Wavefront*



A notação  $P_i^k$  significa o trabalho do processador  $i$  na rodada  $k$ . Assim, inicialmente  $P_1$  começa a computar na rodada 0.

## O algoritmo *Wavefront*

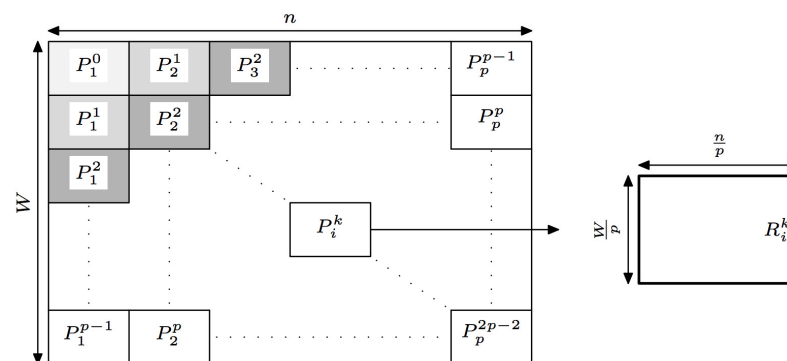


Então  $P_1$  e  $P_2$  podem trabalhar na rodada 1;  $P_1$ ,  $P_2$  e  $P_3$  na rodada 2, e assim sucessivamente.

## O algoritmo *Wavefront*

Em outras palavras:

- Após a computação da  $k$ -ésima parte da sub-matriz  $f_i$  (denotada por  $f_i^k$ ), o processador  $P_i$  envia para o processador  $P_{i+1}$  os elementos da fronteira direita (coluna mais a direita) de  $f_i^k$ . Estes elementos são denotados por  $R_i^k$



Algoritmo: Mochila 0-1 paralelo

## Algoritmo: Mochila 0-1 paralelo

### Entrada

- ▷ O número  $p$  de processadores;
- ▷ O número  $i$  do processador, onde  $1 \leq i \leq p$ ;
- ▷ A capacidade da mochila  $W$ ;
- ▷ Subvetores  $v_i$  e  $w_i$  de tamanho  $\frac{n}{p}$ .

### Saída

- ▷  $f(r, c) = \max\{f(r - 1, c), f(r - 1, c - w_r) + v_r\}$  onde  $1 \leq c \leq W$   
e  $(j - 1)\frac{n}{p} + 1 \leq r \leq j\frac{n}{p}$

## Algoritmo: Mochila 0-1 paralelo

### Algoritmo

**para**  $1 \leq k \leq p$  **faça**

**se**  $i = 1$  **então**

**para**  $(k-1)\frac{W}{p} + 1 \leq r \leq k\frac{W}{p}$ ,  $1 \leq c \leq \frac{n}{p}$  **faça**

*compute*  $f(r, c)$ ;

*envia*( $R_i^k, P_{i+1}$ )

**se**  $i \neq 1$  **então**

*recebe*( $R_{i-1}^k, P_{i-1}$ );

**para**  $(k-1)\frac{W}{p} + 1 \leq r \leq k\frac{W}{p}$ ,  $1 \leq c \leq \frac{n}{p}$  **faça**

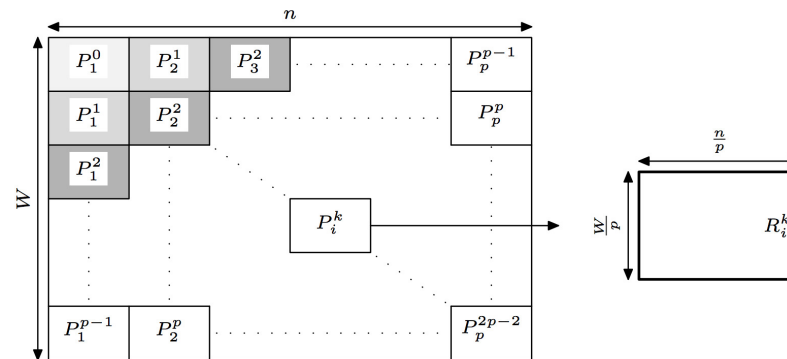
*compute*  $f(r, c)$ ;

**se**  $i \neq p$  **então**

*envia*( $R_i^k, P_{i+1}$ )

Problema da mochila 0-1  
(versão 2)

## Problema da mochila 0-1



É fácil verificar que o processador  $P_p$  só inicia seu trabalho quando o processador  $P_1$  termina a sua computação, na rodada  $p - 1$ . Portanto, temos um balanceamento de carga ruim.

Visando uma melhor distribuição de carga, tentamos fazer com que cada processador inicie o seu processamento o quanto antes.



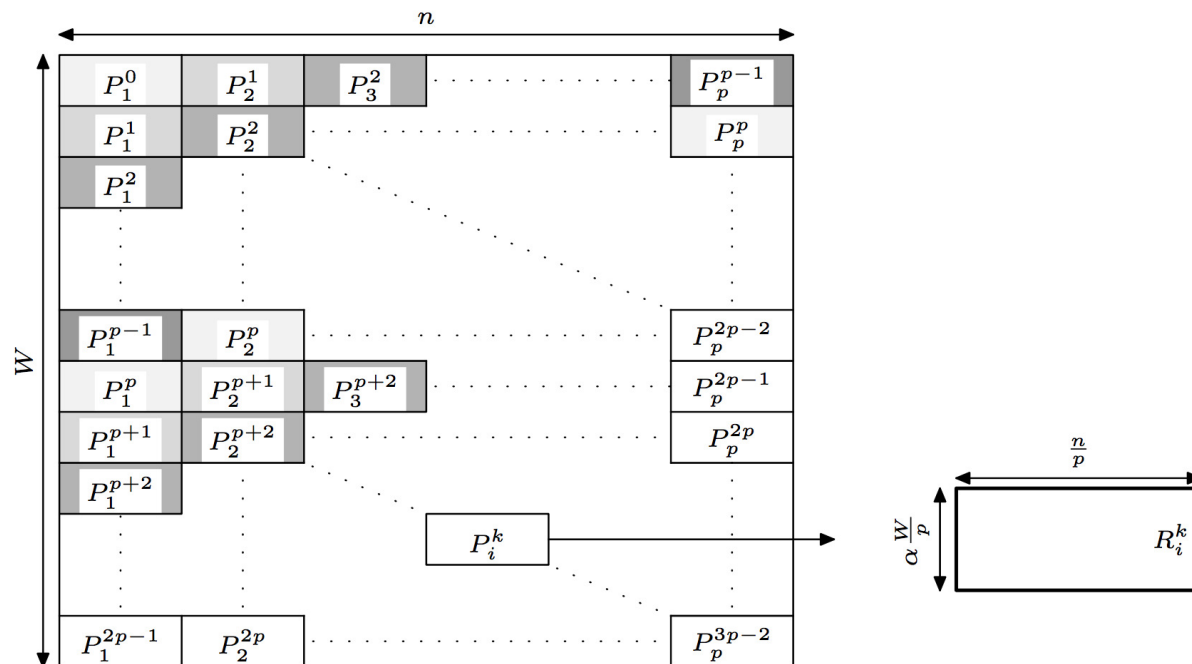
Isso pode ser feito diminuindo o tamanho da mensagem que o processador  $P_i$  envia para o processador  $P_{i+1}$ . Em vez de considerarmos mensagens de tamanho  $\frac{W}{p}$ , consideramos mensagens de tamanho  $\alpha \frac{W}{p}$  e testamos vários tamanhos de  $\alpha$ .

## Algoritmo: Mochila $\alpha$ -0-1 paralelo

### Algoritmo

```
para  $1 \leq k \leq \frac{p}{\alpha}$  faça
  se  $i = 1$  então
    para  $\alpha(k-1)\frac{W}{p} + 1 \leq r \leq \alpha k \frac{W}{p}, 1 \leq c \leq \frac{n}{p}$  faça
      compute  $f(r, c)$ ;
      envia( $R_i^k, P_{i+1}$ )
  se  $i \neq 1$  então
    recebe( $R_{i-1}^k, P_{i-1}$ );
    para  $\alpha(k-1)\frac{W}{p} + 1 \leq r \leq \alpha k \frac{W}{p}, 1 \leq c \leq \frac{n}{p}$  faça
      compute  $f(r, c)$ ;
    se  $i \neq p$  então
      envia( $R_i^k, P_{i+1}$ )
```

## Problema da mochila $\alpha$ -0-1



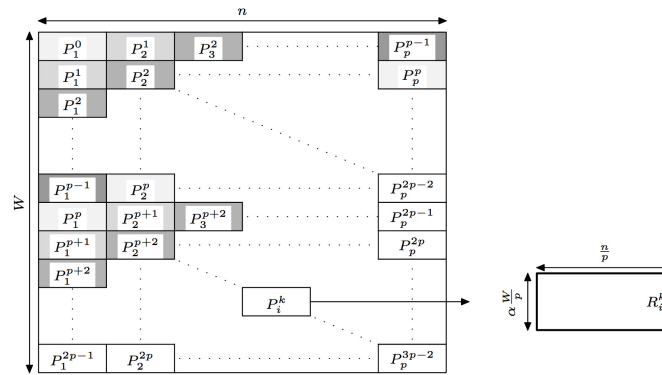
Um escalonamento de  $O(p)$  rodadas de comunicação e  $\alpha = 1/2$

## Problema da mochila 0-1

**Teorema 1.** *O algoritmo utiliza  $O(\frac{Wn}{p})$  computação local com  $(1 + \frac{1}{\alpha})p - 2$  rodadas de comunicação.*

**Prova.** O processador  $P_1$  envia  $R_1^k$  para o processador  $P_2$  após computar o  $k$ -ésimo bloco  $\frac{\alpha W}{p}$  de linhas da  $\frac{Wn}{p}$  submatriz  $f_1$ . Após  $\frac{p}{\alpha} - 1$  rodadas de comunicação, o processador  $P_1$  termina o seu trabalho.

Similarmente, o processador  $P_2$  encerra o seu trabalho após  $\frac{p}{\alpha}$  rodadas de comunicação. Assim, depois de  $\frac{p}{\alpha} - 2 + i$  rodadas de comunicação, o processador  $P_i$  finaliza seu o trabalho.



Visto que temos  $p$  processadores, após  $(1 + \frac{1}{\alpha})p - 2$  rodadas de comunicação, todos os  $p$  processadores terão acabado seu trabalho.

Cada processador utiliza um algoritmo de programação dinâmica sequencial para computar a solução ótima da submatriz  $f_i$  para o problema da Mochila 0-1. Consequentemente este algoritmo utiliza tempo  $O(\frac{Wn}{p})$  de computação local em cada processador  $p$ . □

## Problema da mochila 0-1

**Teorema 2.** *No final do algoritmo,  $f(n, W)$  encontraremos a solução ótima para o Problema da Mochila 0-1 com  $n$  itens, valores  $v_i$  e pesos  $w_i$ ,  $1 \leq i \leq n$  e capacidade  $W$ .*

**Prova.** O Teorema 1 prova que após  $(1 + \frac{1}{\alpha})p - 2$  rodadas de comunicação, o processador  $P_p$  termina seu trabalho. Essencialmente, estamos computando um algoritmo de programação dinâmica sequencial para o Problema da Mochila 0-1 e enviando as fronteiras para o processador da direita, a corretude do algoritmo aparece naturalmente com a corretude do algoritmo sequencial.

Portanto, após  $(1 + \frac{1}{\alpha})p - 2$  rodadas de comunicação,  $f(n, W)$  armazenará a solução ótima para o Problema da Mochila 0-1 com  $n$  itens, valores  $v_i$  e pesos  $w_i$ ,  $1 \leq i \leq n$ , capacidade  $W$ .  $\square$

Fim