ISSN 0101 - 9201

CBPF-NT-001/87

O CONTROLE DE ALOCAÇÃO DE MEMÓRIA EM REDUCE[1]

por

Renato Pires dos Santos

Centro Brasileiro de Pesquisas Físicas - CBPF/CNPq
Rua Dr. Xavier Sigaud, 150
22290 - Rio de Janeiro, RJ - Brasil

[1]Baseado em material a ser concluído no nosso livro "Introdução ao sistema REDUCE de cálculo algébrico" (em preparação).

RESUMO

Procuramos aqui fornecer informações úteis aos usuários de REDUCE quanto à alocação de memória, de forma a permitir não só uma menor exigência ao computador, como também a realização de cálculos maiores com a mesma quantidade de memória total.

Palavras-chaves: Computação algébrica; Sistema REDUCE; Alocação de memória.

1 Introdução

Qualquer programa se expandirá até ocupar toda a memória disponível.

As Leis de Murphy

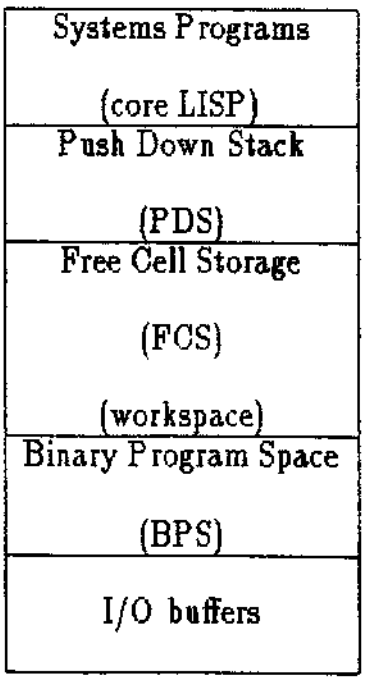
A grande maioria dos usuários de REDUCE já teve a desagradável experiência de ter um cálculo envolvendo expressões grandes interrompido com uma mensagem de erro indicando necessidade de mais memória que a disponível.

Também é fato que os usuários de Computação Algébrica são frequentemente responsáveis por sobrecargas nos sistemas, sendo por isso considerados algo antisociais em alguns CPD's.

Nas seções seguintes pretendemos esclarecer ao usuário comum de REDUCE (que por vezes não tem acesso aos manuais de implementação) a origem de seus problemas e fornecer informações que lhe permitam uma melhor convivência com o sistema.

2 A estrutura de memória

O sistema REDUCE é uma aplicação de LISP (mais especificamente, SLISP) e, por conseguinte, a sua estrutura interna de espaço de memória reflete a de LISP, a qual é indicada no diagrama abaixo



System programs Contém:

- o interpretador
- funções LISP predefinidas para controle do programa (core LISP)
- o coletor de lixo
- o tratador de erros

Free Cell Storage Contém estruturas de dados na forma de listas de células de 64 bits (8 bytes), chamadas *palavras* ou *células de* FCS. Quando todas as células são utilizadas, o coletor de lixo é acionado.

Push Down Stack Contém endereços e estruturas de dados, especialmente para recursão. Inicialmente, consiste em 10000 células.

Binary Program Space Contém funções LISP compiladas (não incluídas no núcleo inicial LISP, armazenadas em *System Programs*). Inclui algumas funções de SLISP e também LAP e o compilador. Usado para funções compiladas de REDUCE e carregadas dinâmicamente (p. ex. BFLOAT, MATR, SOLVE, etc.) e também para novas funções compiladas.

I/O buffer : área de memória reservada ao sistema para uso em I/O, na forma de *buffer*. Inicialmente monta a 8Kbytes.

3 A alocação inicial de memória

REDUCE usará toda a memória disponível, a menos que seja limitado. O parâmetro `C=`*nnnn* (*core*) limita a quantidade de memória durante a instalação a *nnnn* bytes (ou `C=`*nK* , a *n* Kbytes ou `C=`*nP* , a *n* páginas de 4 Kbytes cada).

A área de trabalho (*workspace*) é automaticamente expandida durante a instalação até preencher toda a memória disponível.

Se nenhum valor é fornecido ao parâmetro `C` ou é muito grande para o ambiente, o valor é ajustado ao máximo disponível. O número que aparece em `SLISP`: no início da sessão é o número de bytes de memória disponível a REDUCE internamente.

Para performance razoável, um mínimo de 750 Kbytes de memória de execução é recomendado. Pode ser apenas 375 Kbytes, embora com este valor muito pouca memória fique disponível para *workspace*.

Com uma área de trabalho maior, cálculos maiores podem ser realizados e o tempo de execução é reduzido pois é necessária menos administração de espaço (recolhimento de lixo menos frequente).

O tamanho do espaço do *I/O buffer* retornado ao sistema (geralmente 8 Kbytes) pode ser controlado pelo parametro `R=`*nnnn*.

O tamanho inicial de PDS é 10000 palavras e pode ser controlada por `S=`*n* durante a instalação .

O espaço interno de REDUCE é dividido entre FCS e BPS. O parâmetro **F=***n* alocará *n*% do espaço a FCS e (100-*n*)% a BPS. O parâmetro **B=***n* alocará *n*% a BPS e (100-*n*)% a FCS. O valor usual é **B=34** .

A instalação padrão de REDUCE comprime FCS a 25000 palavras e reserva 18000 palavras para BPS[1].

4 Otimizando a alocação de memória

Uma vez que as funções compiladas são carregadas dinamicamente, quando referidas pela primeira vez numa sessão, não é necessário reservar BPS suficiente para carregá-las todas. Só é preciso garantir espaço para as funções que estarão carregadas simultaneamente.

Pode-se saber quais módulos estão carregados com o comando

```
SYMBOLIC MLIST!*;
```

Durante o carregamento dinâmico de algumas funções, podem aparecer mensagens da forma

```
*** LOGAND REDEFINED
```

que, no entanto, podem ser ignoradas.

Se a mensagem

```
*** PUSHDOWN STACK OVERFLOW
```

ocorre, significa que foi estabelecida uma recursão infinita ou que o problema é tão complicado que a capacidade de pilha (*stack*) foi excedida. Se nada errado foi tentado, o tamanho da pilha pode ser aumentado com o comando **CONDENSE** (vide adiante detalhes deste comando).

O espaço reservado para BPS não é disponível para FCS e o decréscimo no *workspace* reflete negativamente na performance do sistema.

O BPS desnecessário pode ser liberado e devolvido a FCS. Isso pode porém frustrar o mecanismo de carga automática se o espaço mantido for insuficiente.

Para saber-se quanto BPS está correntemente sem utilização, usa-se o comando

```
SYMBOLIC BPSLEFT;
```

Para liberar BPS, retornando espaço ao *workspace*, usa-se

```
END;
(BPSMOVE nnnn)
(BEGIN)
```

[1] A opção **PAR=BIG** da implementação de REDUCE no Laboratório Nacional de Computação Científica (LNCC) reserva 45000 para BPS.

onde *nnnn* é o número de palavras de BPS a serem mantidas disponíveis (vide adiante as necessidades de cada módulo quanto a BPS para se estimar o valor adequado).

Se a mensagem

***** GO-STORAGE EXHAUSTED**

ocorre, o problema tratado exige mais FCS do que disponível. Deve-se tentar reexecutar o programa com mais memória (parâmetro **C** com valor maior[2]) ou liberar BPS não utilizado a FCS com o comando **BPSMOVE**.

Quando o BPS alocado for insuficiente para a função que se deseja carregar, aparecerá a mensagem

***** (BPSMOVE** *nnnn*) **AFTER CONDENSE**

Neste caso, deve-se usar

```
SYMBOLIC CONDENSE (ssss,fff);
(BPSMOVE nnnn)
(BEGIN)
```

O comando **CONDENSE** definirá o tamanho da pilha como *ssss* palavras, tentará comprimir as listas em FCS às *fff* células mais próximas a PDS, executando coleta de lixo.

BPSMOVE moverá a área de programas binários para longe de FCS, mantendo *nnnn* palavras do limite, liberando células livres a FCS . O número retornado é o número de bytes pelos quais BPS foi deslocado (8 vezes o número de palavras liberadas a FCS).

Se novas funções estão sendo compiladas e a mensagem

***** BPS FULL**

ocorre, a sequência acima pode ser utilizada.

5 Necessidades de memória dos vários módulos

Os vários módulos têm as seguintes necessidades

[2] Opção **PAR=BIG**, no caso do LNCC.

função	nome	BPS	FCS
operações matriciais	MATR	2000	210
números c/ precisão arb.	BFLOAT	8500	2930
solução de equações	SOLVE	4300	2780
fis. de altas energias	HEPHYS	3700	590
integrador	INT	14000	3770
fatorizador	FACTOR	31700	8640
compilador	CMACRO,		
(três módulos)	COMPLR,		
	LAP	9160	4000
versões carga rápida	FAP	3200	800

Vê-se, portanto, que a reserva de 18000 palavras de BPS não é suficiente para carregar o módulo FACTOR, embora o seja para vários outros.

6 Conclusões e Agradecimentos

A técnica apresentada acima permite que muitos cálculos aparentemente impossíveis de serem realizados numa dada implementação, devido ao seu tamanho, possam agora ser realizados nessa mesma implementação. Exemplo disso é o trabalho de Srivastava[8].

O usuário interessado em outras informações sobre a implementação de REDUCE deve consultar os manuais *REDUCE user's guide* e *REDUCE installation guide* para as máquinas correspondentes. Estes manuais vêm na fita de distribuição de REDUCE e devem ser disponíveis nos CPD's onde REDUCE estiver implementado.

Agradeço a P. P. Srivastava pelo estímulo inicial em estudar como tornar tratáveis certos problemas em REDUCE pela otimizaçãao da alocação de memória.

[8] P. P. Srivastava, *Truncated Compound Poisson (TCP) alternative to multiplicity distribution in hadron-hadron collision*, preprint CERN- TH.4755/87