Uma Avaliação da Expressividade de Paralelismo da Linguagem Actus

L.E. Favre [1] C.L. de Amorim [2]

RESUMO

A linguagem de programação Actus foi projetada originalmente para se programar máquinas vetoriais (*vector and array processors*) oferecendo um modelo de programação paralela síncrona com espaço de endereçamento global.

Neste trabalho a expressividade de paralelismo da linguagem Actus é avaliada utilizando 100 loops construídos especialmente para testar a eficiência de vetorizadores comerciais.

Os resultados revelam que Actus é insuficiente para expressar o paralelismo de todos os loops. Uma simples extensão da linguagem é proposta , tornando possível expressar o paralelismo de 99% dos loops. Esse resultado é contrastado com os 72% conseguidos automaticamente pelos melhores vetorizadores comerciais.

ABSTRACT

The Actus programming language has been originally designed to program vector and array processors by offering a synchronous parallel programming model and a global addressing space.

In this work, the expression of parallelism of Actus is evaluated by using a test suit of 100 loops which has been specially built to test the efficiency of commercial vectorizers.

The results so far show that Actus is unsufficient to express all parallelism of the loops. However a minor extension is proposed to Actus which allows 99% of all loop parallelism to be expressed. This result contrast to the 72% obtained automatically by the best commercial vectorizers.

[1]MSc (COPPE - 1992), áreas de interesse : Processamento Paralelo (arquiteturas, linguagens de programação e compiladores)

[2]MSc (COPPE - 1979), PhD (Imperial College - 1984); áreas de interesse: Supercomputação e Processamento Paralelo; Professor Adjunto da COPPE/Sistemas, UFRJ.
COPPE/Universidade Federal do Rio de Janeiro
Caixa Postal 68511 - CEP 21945-970 - Rio de Janeiro - RJ
Email amorim @ rio.cos.ufrj.br

* Esse trabalho foi parcialmente financiado pelo CNPq.

1 Introdução

Na área de computação científica computadores paralelos vem sendo utilizados em busca de tempos de execução dos programas cada vez menores, apesar da maior complexidade de programação e do maior tempo de desenvolvimento das aplicações.

Embora várias propostas de linguagens de programação paralela existam, o uso da linguagem Fortran 77 e dialetos com extensões para programação paralela continuam predominantes. Neste caso, para se obter maior desempenho recorre-se ao uso de compiladores que reestruturem o programa seqüencial para que as operações paralelas ou vetoriais oferecidas pela máquina alvo possam ser aplicadas. Uma outra alternativa é a linguagem de programação Actus [1].

A proposta de Actus oferece ao programador uma linguagem Pascal-like [5] com mecanismos que permitem descrever estruturas de dados paralelas e operações sobre estes dados. É função do compilador [7] [9] a partir do paralelismo expresso no programa fonte, utilizar o máximo de paralelismo permitido pela máquina alvo.

A linguagem Actus foi projetada para se programar máquinas vetoriais, oferecendo um modelo de programação paralela síncrona, com espaço de endereçamento global. Contrastando, por exemplo, com à proposta recente do HPF (High Performance Fortran) [12] que é orientada também para máquinas paralelas, Actus não possui notação para a decomposição e para a distribuição de dados entre os processadores. Semelhantemente porém, Actus e HPF ao oferecerem ao programador construções paralelas, simplificam a compilação e tornam mais eficiente a geração de código para máquinas paralelas.

Neste trabalho a expressividade de paralelismo da linguagem Actus é avaliada. Para isso uma coleção de 100 loops Fortran apresentados em [3], abrangendo uma área significativa de aplicações, foram utilizados. Descrevemos esses loops em Actus, visando explicitar o paralelismo existente nestes loops e avaliar a capacidade de Actus para esse fim. Essa coleção compreeende um número expressivo de testes fundamentais para se avaliar a capacidade de vetorização automática dos compiladores otimizadores existentes de máquinas vetoriais e paralelas comerciais [3].

Através deste estudo, verificamos que o paralelismo de Actus permite expressar um número significativo dos loops, mas é insuficiente para todos. A notação de Actus é estendida e através dela conseguimos expressar o paralelismo de 99 loops, contrastando com os 72 conseguidos pelo compiladores vetorizadores.

Este trabalho é organizado como se segue. Na seção 2 é feita uma análise do modelo de paralelismo de Actus. Na seção seguinte uma descrição dos experimentos e a eficiência dos compiladores vetorizadores utilizados são apresentadas. A expressividade do paralelismo da linguagem extendida é avaliada na seção 4, e conclusões parciais são observadas na seção 5.

2 O Modelo de Paralelismo Síncrono de Actus

Actus tem por objetivo fornecer ao programador uma linguagem onde possa confortavelmente expressar o paralelismo mais imediato do algoritmo, e com isso aumentar a clareza do programa, além de reduzir o esforço de programação quando comparado com uma linguagem seqüencial.

Outro objetivo é o de que as construções paralelas possam ser diretamente e eficientemente compiladas para a máquina alvo. Entretanto, Actus não está ligada a nenhum tipo de arquitetura sendo possível que algumas das construções paralelas não possam ser exploradas pela máquina alvo, sendo então tratadas como seqüenciais. O compilador deve também através da análise de dependência de dados e de controle [8], reestruturar o código, objetivando a execução de um maior número de comandos em paralelo (além dos definidos pelo programador).

Actus estende a linguagem Pascal com a inclusão de dados paralelos, e construções que facilitam o processamento paralelo. Um resumo sucinto destas extensões é apresentado a seguir:

1. Introdução de uma notação especificando quais matrizes que podem ser manipuladas em paralelo.
2. Inclusão de constantes paralelas, e meios de especificar em um conjunto de dados (matriz) quais serão tratados em paralelo (conjunto de índices).
3. Introdução do comando USING, cujo corpo define as partes do programa que contém referências paralelas.
4. Introdução dos operadores SHIFT e ROTATE para alinhamento dos dados paralelos referenciados. Alteração das funções para que possam retornar matrizes paralelas.
5. Alteração das funções padrão de modo que possam aceitar como parâmetros matrizes paralelas e onde aplicável, retornar com matrizes paralelas.
6. Método pelo qual matrizes paralelas de tamanho variável podem ser passadas como parâmetros para procedimentos e funções.

Actus é mais adequada à expressão do paralelismo de dados. A notação adotada previlegia a expressão do paralelismo temporal, mais precisamente o paralelismo explorado por processadores vetoriais *pipelined*. Todavia, array de processadores operando em lock-step permitem que o paralelismo expresso por Actus possa ser explorado espacialmente. Considere o seguinte exemplo.

```
1  USING IS:=1:100 DO
2    A[IS] := B[IS] * 5.
3    B[IS] := C[IS] + D[IS];
4  END
```

Num comando de atribuição dentro do comando USING (linhas 2 a 3) a expressão à direita do sinal de atribuição é avaliada para todos os índices definidos pelo conjunto de índices (IS). A atribuição do valor calculado a variável à esquerda do sinal de atribuição é então feita simultaneamente para todos os índices. Somente após o término de todas as operações envolvendo o comando da linha 2, o comando da linha 3 poderá ser executado.

Todo o paralelismo em Actus é explorado dentro dos comandos USING. Esta restrição leva o programador a adotar em Actus um estilo de programação onde os comandos paralelos fiquem agrupados segundo uma mesma extensão de paralelismo, preservando a característica

de programação disciplinada de Pascal. Actus permite, dentro de um comando USING definir operações paralelas de várias dimensões, definir índices com espaçamento diferente de 1, e índices irregulares. A indexação pode ser direta ou indireta, e operadores podem ser aplicados aos índices, alterando o alinhamento entre eles, através dos operadores SHIFT e ROTATE.

A notação de Actus para arrays paralelos é semelhante à adotada por outras linguagens paralelas [4] [12], existindo uma certa semelhança entre o comando USING de Actus com o FORALL de vários dialetos paralelos do Fortran e adotado pelo HPF.

3 Descrição dos Experimentos

Com a finalidade de testar a eficácia de compiladores vetorizadores Fortran comerciais, uma coleção de 100 loops foi selecionada por pesquisadores em compiladores vetorizadores. Os loops testam características específicas dos compiladores. Estes loops refletem construções cuja vetorização variam da mais simples à extremamente difícil. Os loops foram submetidos a compiladores vetorizadores comerciais, e para uma variedade de máquinas alvo, incluindo supercomputadores, mini-supercomputadores e mainframes.

3.1 Características dos Loops

O objetivo dessa massa de teste é o de avaliar a eficiência dos compiladores em quatro grandes áreas: análise de dependência, vetorização, reconhecimento do idioma (Fortran) e completitude da linguagem. Dentro destas áreas várias sub-categorias existem. Os loops estão distribuídos nestas quatro áreas da seguinte forma:

1. Análise de Dependência. São 24 os loops nesta categoria, rotulados de S111 até S175[3].
2. Vetorização. São 34 os loops nesta categoria, de S211 até S293.
3. Reconhecimento do Idioma. São 15 os loops nesta categoria, de S311 até S342.
4. "Language Completeness". São 27 os loops nesta categoria, de S411 até S4117.

Análise de Dependências

A análise de dependências compreende duas áreas: análise de fluxo de dados globais e teste de dependência. É a principal área de atuação dos vetorizadores. Algumas das sub-áreas são: Teste de dependência linear, Reconhecimento da variável de indução, Análise global do fluxo de dados, Teste de dependência não linear, Analise de fluxo de dados inter-procedurais, Fluxo de controle e Variáveis simbólicas.

Vetorização

Loops DO de um só comando podem ser fácilmente vetorizáveis. Para construções mais complexas são necessárias técnicas mais sofisticadas onde o compilador reestrutura o código

[3]Esta numeração não é contígua

para permitir a vetorização. Algumas da sub-áreas são: Reordenação de comandos, Distribuição de loops, Troca de loops, Desmembramento de Nodo, Expansão de matriz e escalar, Renomeação de escalar, Fluxo de controle, Ultrapassagem de limites, Desenrolamento de loops e Vetorização de acesso a diagonais.

Reconhecimento do Idioma

Reconhecimento de idioma se refere à identificação de formas particulares que permitem implementações especiais (possívelmente mais rápidas). Algumas das sub-áreas são: Reduções, Recorrências, Loops de busca, Empacotamento.

Completitude da Linguagem

Se refere a quão efetivamente o compilador compreende toda a linguagem Fortran. Compiladores mais simples podem se limitar à análise de loops DO contendo apenas atribuições de inteiros e reais. Compiladores mais sofisticados analisam todos os loops e irão vetorizar sempre que possível. Algumas das sub-áreas são: Reconhecimento de loops, Classe de armazenamento e equivalência, Parâmetros, IF não lógico, Funções intrínsecas, Comandos de entrada e saída, GO TO não locais, Semântica de vetores, Tipos de dados, Endereçamento indireto e Comandos com chamada de funções.

3.2 Eficiência dos Vetorizadores

Metodologia de Teste

Os loops foram escritos em Fortran 77 [11] e compilados sem sofrer o acréscimo de nenhuma diretiva de compilação. Foi utilizada apenas a opção de vetorização automática dos compiladores.

Contagem dos Loops

Foram considerados três tipos de resultados na vetorização de um loop. Um loop é considerado vetorizado se o compilador gerar instruções vetoriais para todos os comandos vetorizáveis do loop. Um loop é considerado parcialmente vetorizável se o compilador gerar isntruções vetoriais para alguns mas não todos os comandos vetorizáveis do loop. Um loop é considerado não vetorizado se o compilador nao gerar instruções vetoriais para qualquer comando vetorizável dentro do loop.

Análise dos Resultados Obtidos

O número médio de loops totalmente ou parcialmente vetorizáveis foi de 59%. O melhor resultado foi de 72% (67% totalmente e 5% parcialmente vetorizado pelo compilador).

4 Expressividade de Paralelismo de Actus

Um estudo da expressividade de paralelismo de Actus foi feita, programando-se nesta linguagem a coleção dos referidos 100 loops em Fortran.

O objetivo é verificar em que medida podemos expressar todo o paralelismo existente nesses loops e se o conjunto de comandos paralelos de Actus é suficiente e adequado (visando a facilidade de programação e clareza do programa).

Máquinas MIMD e SIMD

Embora as máquinas vetoriais ainda predominem na área de computação de alto desempenho, este espaço vem sendo gradativamente ocupado pelas máquinas paralelas que oferecem um paralelismo tipo MIMD, (máquinas vetoriais oferecem um paralelismo tipo SIMD ou MSIMD).

Contudo, a maioria dos programas utilizados em ambos os tipos de máquinas são escritos em Fortran 77, e não há perspectiva a curto prazo de mudança deste quadro, mesmo com a eventual predominância das máquinas MIMD com centenas de processadores referidas como MMP. Esforços tem sido feitos pela comunidade para tornar a linguagem Fortran mais adequada a essas máquinas com a definição de um novo padrão (Fortran-90 [13]) e a definição de uma extensão sugerida pelo HPF para que aquela linguagem possa fornecer suporte à programação das MPP.

Mesmo com o advento deste novo paradigma na computação paralela, os compiladores continuarão a utilizar as técnicas de otimização e reestruturação de código endereçadas por aqueles 100 loops.

Actus cuja proposta original data de 1979 [2], conceitualmente ataca os mesmos problemas do HPF, variando bastante contudo na forma como isto é feito. Actus não aborda o problema da distribuição de dados, enquanto que o HPF permite que o programador ofereça diretivas indicando como os dados deverão ser distribuídos. Actus oferece uma extensão à linguagem Pascal, onde o comando USING de Actus se assemelha ao comando FORALL de HPF e a indexação de arrays paralelos de Actus se aproxima a utilizada por Fortran 90 e também a de HPF.

Nosso interesse é utilizar este estudo [3] para quantificar a capacidade de Actus em exprimir o paralelismo potencial existente no programa. Consideramos assim que estes loops representam uma parcela significativa embora não exaustiva dos comandos necessários a descrição de algoritmos paralelos.

Uma linguagem de programação paralela deve contudo endereçar outros problemas como eficiência do código gerado, clareza do programa, transportabilidade do código, facilidade de programação, ser poderosa o suficiente para expressar todo o tipo de paralelismo vislumbrado pelo programador (ou existente no algoritmo). Neste trabalho estamos portanto enfocando primordialmente um destes aspectos. Como Actus é uma linguagem muito simples (comparada com Fortran 90 por exemplo) e como uma parte significativa do paralelismo é explicitado pelo programador, o compilador pode ser mais simples e se dedicar a outros problemas relacionados a melhoria do desempenho do programa, p.ex., a distribuição dos dados.

Metodologia

O critério adotado para transcrever os loops para Actus, foi o de manter a compatibilidade semântica entre os dois, e o de usar a notação paralela onde possível, note que estes loops foram transcritos manualmente, com interesse específico de verificar a expressividade de Actus em uma gama variada de aplicações. Adotamos a mesma definição de [3] para caracterizar os loops como vetorizados, parcialmente vetorizados e não vetorizados.

No estudo destes loops constatou-se que com um pequeno conjunto de modificações é possível tornar Actus mais versátil e poderosa. Num trabalho anterior [10] foi proposta uma alteração de Actus visando prioritariamente a facilidade de compilação, programação e depuração da linguagem além de uma maior transportabilidade.

Aqui o enfoque recai sobre a expressividade de Actus, sendo algumas extensões à linguagem apresentadas, por meio das quais um maior número de loops podem ser escritos utilizando a notação paralela. Preservando a simplicidade original de Actus estas alterações se resumem em dar um maior grau de liberdade ao programador na utilização dos conjuntos de índices.

Em Actus, a definição dos conjuntos de índices sobre os quais as operações paralelas serão executadas é muito rígida, e só permite um conjunto limitado de operações sobre os mesmos, utilizando os operadores SHIFT e ROTATE. Permitindo um número maior de operações sobre os índices, (semelhantes às aceitas pelo HPF) aumenta-se a versatilidade do comando USING e expande-se o conjunto de operações paralelas que podem ser descritos por Actus.

Os 100 loops foram transcritos para Actus em duas etapas. Na primeira utilizou-se apenas a notação original de Actus, com a qual alguns dos loops foram descritos utilizando somente a notação seqüencial, não sendo possível utilizar nenhum comando paralelo.

Na segunda etapa utilizou-se a linguagem extendida, através da qual foi possível expressar em paralelo grande parte dos loops que anteriormente não puderam ser descritos com a notação paralela. A utilização da notação paralela além de facilitar o esforço de programação torna o programa mais legível e simplifica o trabalho de compilação.

Resultados Obtidos

Dos 100 loops, transcritos para Actus obtivemos os seguintes resultados: [4]

1. Em 10 loops, não foi possível utilizar a notação paralela de Actus, sendo descritos utilizando-se somente a notação seqüencial.

2. Em 6 loops, só foi possível utilizar a notação paralela em uma parcela dos comandos do loop, e para os demais comandos do loop a notação seqüencial foi utilizada.

3. Os 84 loops restantes foram totalmente paralelizados utilizando os atuais recursos da linguagem.

4. Dentre estes 84 loops, 15 loops foram paralelizados utilizando também funções de redução da biblioteca de Actus.

Os motivos que impediram que os 16 loops pudessem ser totalmente paralelizados foram analisados, constatando-se que com pequenas alterações Actus pode paralelizar 15 loops. As alterações sugeridas são descritas a seguir. Na definição deste pequeno grupo de sugestões para extensão de Actus, buscou-se soluções que preservassem a simplicidade da linguagem, que pudessem ser incorporadas naturalmente por Actus sem descaracterizá-la, sem onerar proibitivamente o esforço de compilação e que possam ser utilizadas de modo intuitivo pelo programador.

[4] Note que nos limitamos a transcrever para Actus apenas o código referente aos loops em cada subrotina, ignorando as declarações de dados e os comandos de saída

4.1 Loops não Paralelizáveis ou Parcialmente Paralelizáveis

Índices Dinâmicos

Foram 10 os loops que não puderam ser expressos utilizando a notação paralela de Actus. Em 6 destes loops a existência de dependência de dados em um mesmo comando de atribuição impede que a notação paralela seja utilizada. São eles: S221, S222, S321, S322, S323 e S411. Uma solução é introduzir em Actus uma variante para a sintaxe do comando USING para estes casos, com a introdução do conceito de índices dinâmicos.

Uma solução seria marcar qual ou quais conjuntos de índices não podem ser tratados em paralelo. Qualquer um dos 6 loops em questão, poderia ser expresso utilizando esta notação, como mostrado abaixo para o fragmento do loop S321, na versão original (Fortran) e em Actus:

```
      subrotine s321
        ...
      do 870 i = 2,n
        a(i) = a(i) + a(i-1)*b(i)
870   continue
        ...

      USING *IS:=2:N DO
        A[IS] := A[IS-1] + A[IS] * B[IS];
      END
```

Note que mesmo com a existência da dependência de dados é possível explorar paralelismo tipo *pipeline* no comando acima, o que pode ser feito pelo compilador com a colocação de barreira de sincronismo relacionando as iterações com relação de dependência.

Também é possivel mesclar índices que podem ser tratados em paralelo com índices que devem ser tratados em seqüência. Exemplo

```
      USING *IS:=2:100, JS:=5:20 DO
        A[IS, JS] [IS-1, JS] + B[IS, JS] * 5
      B[IS, JS] := C[IS, JS] + D[IS, JS];
END
```

Neste exemplo, o * no índice IS indica que as linhas das matrizes deverão ser tratadas uma por vez, a primeira iteração utilizando IS = 2 e a última para IS = 100. As colunas no entanto poderão ser acessadas em paralelo. Adotando-se esta variação do USING, os seis loops que foram parcialmente vetorizados puderam ser expressos sem utilizar a notação seqüencial.

Múltiplos Índices por Dimensão

Dentre os loops que não puderam ser totalmente paralelizados, em oito deles o motivo foi o de operações mais complexas com a variável de iteração (utilizada como índice) que não podem ser descritas em Actus utilizando-se de modo convencional o conjunto de índices. Estes loops são: S122, S126, S127, S171, S281, S452, S4114 e S4115.

Na definição do comando USING, o número de conjunto de índices declarados determina a dimensão do comando. A declaração em uma mesma dimensão de mais de um conjunto de índices (sempre com o mesmo número de elementos) permite que operações com conjunto de índices com espaçamento diferentes possam ser facilmente descritas.

O uso de { } indica que todos os conjuntos de índices assim agrupados pertencem a uma mesma dimensão.Apresentamos como exemplo o loop S126.

```
      subrotine s126(a,bb,n)
        ...
      do 40 i = i,n
        do 41 j = 2,n
          bb(i,j) = bb(i,j-1) + a(k)
          k = k+1
41      continue
      k = k+1
40    continue
        ...

      FOR J := 2 TO N DO
        USING {IS:=1:N; KS:=J:[N]N*N+J} DO
          BB[IS,J] := BB[IS,J-1] + A[KS];
        END
      END
```

A utilização de múltiplos índices por dimensão permitiu que todos os sete loops mencionados acima pudessem ser totalmente paralelizados. Em alguns casos a declaração dos conjuntos de índices não fica muito legível, aumentando as chances de ocorrerem erros de programação na definição das condições de contorno. No exemplo acima isto pode ser observado na definição do limite superior do conjunto de índices KS. Algumas sugestões são apresentadas a seguir com o intuito de facilitar a declaração desses índices.

Operações com os Conjuntos de Índices

A definição de um conjunto de índices é feita da seguinte forma:

INÍCIO:[STEP]FIM;

onde início define o valor do primeiro índice, STEP o espaçamento entre os valores subseqüente dos índices e FIM o limite superior para o último índice.

Contudo há casos (como no loop S126) em que o número de índices (a cardinalidade do conjunto de índices) é o valor que se possui, sendo então o valor de FIM sendo representado por uma expressão, aumentando desnecessariamente o esforço de programação e prejudicando a clareza do programa.

A adoção de um segundo modo de definição para o conjunto de índices como:

INÍCIO:[STEP]{ TAMANHO }

tornará em alguns casos a programação mais confortável como no caso do loop S122 apresentado a seguir. Neste mesmo loop S122 é utilizada a função "TAM", uma decorrência natural deste novo modo de definir um conjunto de índices.

A definição do conjunto de índices utilizando-se o número de elementos do conjunto e o uso de múltiplos índices sugere a possibilidade de se definir o tamanho de um conjunto conforme o tamanho de um outro conjunto. Isto pode ser feito através de uma função que determina o tamanho do conjunto de índices que receber como parametro. Aqui adotamos representar esta função por TAM(INDEX_SET).

```
      subrotine s122(xx,yy,n)
        ...
      do 60 i = n1,100,n3
        xx(i) = yy(i)*zz(100-k1)
        k = k+j
60    continue
        ...

      USING {IS:=1:N; JS:=100-K+1:[J](TAM(IS)) DO
        XX[IS] := YY[IS]*ZZ[JS];
      END
```

Podemos ainda permitir que o alinhamento de índices que originalmente é feito por meio dos operadores SHIFT e ROTATE, possam ser feito pelos operadores $+$, $-$ e $*$. A utilização destes operadores permite que os conjuntos de índices possam sofrer ações de alinhamento mais sofisticadas que não poderiam ser feitas utilizando-se apenas os operadores SHIFT e ROTATE. Tal fato é ilustrado no exemplo abaixo do loop S171. Note que a utilização dos operadores $+$ e $-$ torna o operador SHIFT supérfulo, sendo necessário manter apenas o operador ROTATE.

```
      subrotine s171(a,b,n)
        ...
      do 1030 i = 1,n
        a(i*n) = a(i*n)+b(i)
1030  continue
        ...

      USING IS:=1:N; DO
        A[IS*N] := A[IS*N]+B[IS];
      END
```

Ainda uma outra alteração em Actus com respeito ao conjunto de índices é sugerida pelo loop S452. Este loop foi paralelizado apenas parcialmente porque utiliza o conjunto de índices em um contexto não previsto por Actus. A utilização do conjunto de índices como uma constante paralela, como mostrada no exemplo abaixo do loop S452 permitiu que este loop fosse completamente paralelizado.

```
      subrotine s452(a,b,c,m)
        ...
      do 1250 i = 1,m
        a(i) = b(i)+c(i)+i
1250  continue
        ...

      USING IS:=1:M; DO
        A[IS] := B[IS]+C[IS]+IS;
      END
```

Funções de Compressão e Expansão

Para ser possível paralelizar os loops S341 e 242 torna-se necessário acrescentar duas funções à biblioteca de Actus, uma para comprimir os dados de um vetor ou matriz segundo a ação de uma máscara (definida pelo contexto) e outra para fazer a ação inversa, ou seja a de expansão do vetor ou matriz. Estas funções são:

1. COMPRESS(DESTINO,FONTE)
2. EXPAND(DESTINO,FONTE)

Na função COMPRESS o vetor FONTE tem seus elementos definidos segundo o conjunto de índices, e provavelmente por uma máscara que atua sobre este mesmo conjunto de índices. Estes elementos são escritos nas primeiras posições do vetor DESTINO, indexadas também pelo conjunto de índices, mas sem sofrer a restrição da máscara. Os dois loops são apresentados a seguir para ilustrar o uso destas funções.

```
      subrotine s341(a,b,j,n)
        ...
      j = 0
      do 900 i = 1,n
        if(a(i).gt.0)then
          j = j+1
          b(j) = a(i)
        endif
900   continue


      USING IS := 1:N DO
        IF A[IS] := 0 DO
          COMPRESS(B[IS], A[IS]);
      END
```

```
      subrotine s342(a,b,j,n)
        ...
      j = 0
      do 910 i = 1,n
        if(a(i).gt.0)then
          j = j+1
          a(i) = b(j)
        endif
910   continue
        ...
      USING IS := 1:N
        IF A[IS] > 0 THEN
        EXPAND(A[IS], B[IS]);
      END
```

Deve-se observar contudo que a inclusão destas funções a biblioteca de Actus só será justificada se comprovar-se que não são utilizadas apenas em aplicações isoladas, sem representatividade. Para comparação apresentamos um resumo das funções básicas da biblioteca de Actus que foram utilizadas para transcrever alguns dos 100 loops.

1. MIN() determina o menor elemento da matriz
2. MAX() determina o maior elemento da matriz
3. SUM() soma dos elementos da matriz
4. PROD() produto dos elementos da matriz
5. FIRSTTRUE() posição do primeiro elemento com valor TRUE
6. LASTTRUE() posição do último elemento com valor TRUE
7. FIRST() valor do primeiro índice no conjunto de índices
8. LAST() valor do último índice no conjunto de índices

O único loop (S123) que não pode ser expresso em paralelo é mostrado abaixo.

```
      subrotine s123(a,b,c,n)
      j = 1
      do 50 i=1,n
        j = j + 1;
        a(j) = b(i)
        if( c(i) .gt. 0 ) then
          j = j + 1
          a(j) = c(i)
        endif
50    continue
      return
      end
```

5 Conclusão

A linguagem Actus original não foi suficiente para expressar o paralelismo contido nos 100 loops testados. Entretanto, com simples extensões na linguagem foi possível termos sucesso em 99% dos loops. Para isso foi introduzido mais flexibilidade nas operações com o conjunto de índices e a identificação de conjuntos que deverão ser tratados seqüencialmente . A linguagem Actus estendida revelou-se mais poderosa, facilitando o trabalho de programação paralela e tornando o programa mais legível.

Para continuar a avaliação e a ampliação do escopo de paralelismo original de Actus, estudos semelhantes ao deste trabalho estão em andamento que permitirão testar a expressividade de Actus numa classe significativa de aplicações paralelas.

Referências

[1] Perrot, R.H., Lyttle, R.W., e Dhillon., P.S., "The Design and Implementation of a Pascal-Based Language for Array Processor Architetures", *Journal of Parallel and Distributed Computing*, 4 (1987), 266-287.

[2] Perrot, R.H., Crookes, D., Milligan, P., "The Programming Language Actus", *Software—Practice and Experience*, (1983), Vol. 13, 305-322.

[3] Callahan, D., Dongarra, J., Levine, D., "Vectorizing Compilers: A Test Suite and Results", *Argonne National Laboratory, Technical Report No. 109*, 3 (1988).

[4] Hänßgen. S.U., Lukowicz, P., Philippsen, M., Tichy, W.F., "The Modula-2* Environment for Parallel Programming", *Proceedings of the Working Conference on Massively Parallel Programming Models*, Belin, Germany, 9 (1993).

[5] K.Jensen and N.Wirth, "Pascal: User Manual and Report", (3th ed) Springer-Verlag, 1985.

[6] Sales, C.L, "Projeto e Implementação de uma Linguagem Intermediária para *Transputer*" , *Tese M.Sc. Engenharia de Sistemas e Computação, COPPE / UFRJ*, 1990.

[7] Sales, C.L., "Projeto e Implementação de uma Linguagem Intermediária para *Transputer*" , *Tese M.Sc. Engenharia de Sistemas e Computação, COPPE / UFRJ*, 1990.

[8] Maciel, P.M.C.P.F., "Otimização de Programas Actus", *Tese M.Sc. Engenharia de Sistemas e Computação, COPPE / UFRJ*, 1991.

[9] Favre, L.E., "Um Compilador para a Linguagem de Programação Paralela Actus", *Tese M.Sc Engenharia de Sistemas e Computação, COPPE / UFRJ*, 1992.

[10] Favre, L.E., Amorim, C.L., Carneiro, M.C.R., Maciel, P.M.C.P.F., "Um Compilador para a Linguagem Híbrida de Programação Paralela C_Actus", *XII Congresso da Sociedade Brasileira de Computação*, 1993.

[11] ANSI X3J3 Commitee, "The Programming Language FORTRAN *American National Standard X3, 91978*, 4 (1978).

[12] "High Performance Fortran Language Specification", Jan. 25, 1993, Version 1.0 DRAFT.

[13] "ISO. Fortran 90", May 1991. [ISO/IEC 1539: 1991]