Um Compilador para a Linguagem Híbrida de Programação Paralela C_Actus

L.E. Favre [1] C.L. de Amorim [2] M.C.R. Carneiro [3] P.M.C.P.F. Maciel [4]

RESUMO

Neste trabalho é proposta uma nova versão da linguagem de programação paralela Actus, na qual é introduzido o conceito de modularização e modificações orientadas para uma maior simplicidade tanto no processo de compilação como na depuração de programas paralelos além de assegurar uma maior transportabilidade da linguagem para máquinas vetoriais e máquinas MIMD.

A implementação do compilador da linguagem híbrida resultante, denominada C_Actus é discutida.

ABSTRACT

This work proposes a new version for the parallel programming language Actus, in which we introduce the concept of modularization and modifications intended to achieve simplicity to both compiling and debugging of parallel programs and to warrant the language wider portability to vector processors and MIMD architectures.

The compiler implementation for the resulting hybrid language, called C_Actus is discussed.

[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.

[3]BSc (Inst. Matemática - 1993), áreas de interesse: Compiladores e Processamento Paralelo.

[4]MSc (COPPE - 1991), áreas de interesse : Processamento Paralelo (arquiteturas, linguagens de programação e compiladores)..

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 e pelo FINEP.

1 Introdução

C_Actus representa uma evolução da linguagem e do compilador Actus [1], ambos tendo sido alvo de estudos anteriores [2] [3] [4]. As características introduzidas na linguagem e no seu compilador tem o objetivo de permitir o estudo das características desejáveis em uma linguagem de programação paralela, facilitar a introdução de técnicas de otimização envolvendo o paralelismo de controle de dados, e o estudo experimental de adeqüação dessas linguagens às arquiteturas paralelas atuais.

Seu compilador está sendo implementado para gerar código para as máquinas paralelas NCPI8 [5] e iPSC 860 [6]. Devido às características de C_Actus é possível gerar código para máquinas seqüenciais tais como: estação de trabalho SUN e IBM-PC compatível.

C_Actus fornece ao programador uma linguagem paralela de alto nível com uma notação que lhe permite explicitar o paralelismo de dados existente no programa. Pelo fato do compilador gerar código objeto em C [7], são fornecidas ao programador todas as funções existentes na biblioteca de C, e torna-se possível transportar um programa fonte (em C_Actus) para vários sistemas computacionais que possuam um compilador C.

Por manter os identificadores utilizados no programa fonte, é possível que o programador utilize os depuradores disponíveis para C na avaliação da correção do programa em C_Actus. O programa pode ser analisado na sua versão seqüencial antes de ser transportado para uma máquina paralela onde os recursos de depuração são mais escassos.

Devido à possibilidade de utilizar funções dos compiladores de C paralelo [8], o paralelismo oferecido por C_Actus (paralelismo SIMD) é estendido às arquiteturas MIMD.

Através da documentação da estrutura adotada para a tabela de símbolos e para a representação intermediária do programa [9], torna-se disponível uma estrutura de dados adeqüada para uma futura introdução de paralelizadores e otimizadores de código.

Apresentamos inicialmente uma breve descrição dos trabalhos relacionados a ambientes de programação paralela. A seguir as construções paralelas da linguagem Actus são descritas. Na seção 4, as alterações e acréscimos que deram origem a C_Actus são apresentadas e na seção 5 são discutidos aspectos básicos relativos a implementação da linguagem. Finalmente, conclusões são extraídas dos resultados obtidos até o presente.

2 Trabalhos Relacionados

Com o desenvolvimento dos processadores vetoriais *pipelined* ou supercomputadores (séries CRAY e NEC), ao longo dos últimos 20 anos, várias técnicas para se explorar o paralelismo têm sido usadas. A vetorização automática de programas seqüenciais é amplamente utilizada até um certo nível de complexidade, após o qual, a interação entre o programador e o vetorizador é imprescindível. Diretivas são comumente fornecidas pelo programador ao compilador para guiá-lo no processo de vetorização.

Mais recentemente, com a crescente disseminação de computadores paralelos MIMD todo esse conhecimento adquirido tem sido, em parte, reaproveitado. A questão central no paralelismo MIMD é o mapeamento dos dados através da memória distribuída, e novamente verifica-se que a interação com o programador é indispensável. Vários trabalhos foram desenvolvidos nas áreas de vetorização e paralelização de código [10] [11] e muitos deles orientados para a distribuição de dados para máquinas paralelas [12] [13] [14].

Estudos recentes de distribuição de dados possuem três enfoques diferentes. No primeiro, o compilador gera automáticamente toda a distribuição dos dados. No segundo, o próprio programador distribui os dados e testa o programa esperando que a distribuição escolhida

tenha sido boa. Também no terceiro o programador faz a distribuição mas usa uma ferramenta auxiliar que lhe mostra a eficiência da sua escolha.

Além desses mecanismos de se gerar programas paralelos baseados em programas seqüenciais há ainda linguagens que já expressam paralelismo como Fortran-90 [16], HPF [17], e Actus. Tanto Fortran-90 como Actus são linguagens que expressam paralelismo temporal ou vetorial. Já o HPF gera programas para máquinas com memória distribuída e com processamento vetorial.

3 A Linguagem Actus

Actus é uma linguagem de programação paralela baseada na linguagem Pascal [18], e foi projetada para operar em ambientes paralelos síncronos tais como processadores vetoriais e matriciais. Actus fornece uma estrutura de dados e comandos para declarar, acessar e manipular dados em paralelo, permitindo ao programador escrever algoritmos paralelos bem estruturados e independentes da máquina alvo.

Adotando uma posição oposta àquela adotada, por exemplo em Fortran-77, onde o paralelismo é extraído exclusivamete pela ação de compiladores vetorizadores, Actus fornece ao programador uma notação que lhe permite definir o paralelismo que julgar adequado à solução do problema. Ao compilador Actus cabe traduzir este paralelismo definido pelo usuário no paralelismo físico permitido pela máquina alvo.

3.1 Conceitos Básicos

O Paralelismo na Estrutura de Dados

Os dados que se pretende manipular em paralelo devem ser declarados na forma de *arrays*. Na linguagem Actus podem ser manipulados em paralelo até duas dimensões desses *arrays* sendo que na declaração dos *arrays* paralelos, os índices que serão acessados em paralelo devem ser declarados como tal, os demais índices são declarados normalmente. [5]

A Extensão de Paralelismo

Denomina-se "extensão de paralelismo" (**edp**) o número de elementos que são operados simultaneamente. Cada dimensão paralela do *array* define implicitamente uma máxima "extensão de paralelismo" (**edp**). Esta **edp** ou **edp**s menores podem ser usadas para a manipulação dos dados do *array* nas expressões e comandos do programa. Quando as variáveis paralelas são usadas em expressões a **edp** deve ser a mesma para todas as variáveis envolvidas na operação.

O Conjunto de Índices

Para permitir o acesso seletivo ou total aos elementos das variáveis paralelas, Actus utiliza uma construção denominada INDEX, que permite definir um conjunto de valores. Este conjunto contém os índices dos elementos que se deseja acessar, e pode ser declarado de dois modos distintos, caracterizando os seguintes tipos para estes conjuntos:

1. Conjunto explícito de índices. Na declaração atribui-se à variável do tipo INDEX valores que permanecem inalterados durante todo o escopo daquela variável INDEX.

[5] Em C-Actus todos os índices são declarados da mesmo forma.

2. Conjunto redefinível de índices. Na declaração a variável do tipo INDEX assume um tipo inteiro (INTEGER, CHAR, ...), seus valores serão definidos mais tarde, possivelmente em tempo de execução.

Os conjuntos de índices só podem receber valores inteiros e sua declaração é feita à semelhança da declaração das variáveis comuns.

As Constantes Paralelas

É possível declarar identificadores para representar uma seqüência de valores constantes que possam ser acessados em paralelo.

As constantes paralelas podem ser usadas para atribuir valores iniciais aos *arrays* paralelos ou serem usadas como termos em operações paralelas. Como ocorre com os índices, as constantes paralelas só podem assumir valores inteiros.

3.2 Os Comandos Paralelos

Os comandos paralelos fornecem os meios de manipulação das constantes e variáveis paralelas. Estes comandos têm associado uma **edp** unidimensional ou bidimensional que determina quantos e quais elementos sofrerão a ação dos comandos. Mantendo a estrutura modular da linguagem, cada **edp** é associada a um bloco que constitui o escopo daquela **edp**. Todos os comandos inclusos naquele bloco possuirão a mesma **edp**.

A associação da **edp** a um bloco de comandos é feita através do comando USING, que tem a seguinte forma

USING *index-specification* **DO** *statement*

onde *index-specification* define a **edp** válida para os comandos de *statement*. Todos os comandos paralelos devem estar contidos em um comando USING. A concentração dos comandos com a mesma extensão de paralelismo em um mesmo bloco tem por objetivo aumentar a clareza do programa além de facilitar sua implementação e torná-la mais eficiente.

O Comando de Atribuição

O comando de atribuição, será paralelo quando a variável à esquerda do sinal de atribuição for paralela. Similarmente, o resultado de uma expressão será paralelo, se pelo menos um dos termos envolvidos for paralelo. Neste comando todos os elementos (variáveis e constantes) envolvidos devem ter a mesma **edp**, esta restrição é também feita para ambos os termos de uma expressão binária. O resultado será escalar, se ambos os termos forem escalares.

Para permitir que um valor escalar possa ser atribuído aos elementos de uma variável paralela e que também possa ser manipulado em expressões paralelas, os escalares adotam, implicitamente, o valor da edp vigente, quando utilizados em um comando de atribuição ou em expressões paralelas. Assim, o valor escalar será replicado em tantos elementos quantos forem o número de índices da **edp**. Os operadores permitidos nas expressões paralelas são os mesmos utilizados nas expressões escalares.

Os Comandos Condicionais Paralelos

Actus reconhece dois comandos condicionais: o comando IF e o comando CASE, sendo que o comando IF pode ter a cláusula ELSE omitida.

IF *boolean_expr* **THEN** *statement*
IF *boolean_expr* **THEN** *statement* **ELSE** *statement*

CASE *expr* **OF**
case_label_list1 : *statement*;
case_label_list2 : *statement*;
...
END

Se a expressão de controle do comando IF for escalar a **edp** vigente não será alterada, e os comandos de *statement*, se forem executados, estarão sob a influência daquela **edp**. Se a expressão de controle for paralela, a mesma será avaliada segundo a **edp** atual e seu resultado fornecerá um vetor ou uma grade de valores TRUE ou FALSE, que será usada como máscara para a **edp** atual, gerando uma nova **edp** a ser usada para os comandos do IF relativos a cláusula **THEN**. Para os comandos da cláusula **ELSE** os valores da máscara são complementados antes de gerar a nova **edp**.

Assim como no comando IF, o comando CASE será tratado de forma diferente se a expressão de controle for ou não paralela. Se a expressão de controle for paralela, a **edp** vigente será distribuída entre os vários ramos do comando CASE, cada qual utilizando uma máscara própria, resultado da comparação da expressão com a lista de valores de cada ramo. Cada ramo é executado em seqüência até que todos os ramos do comando CASE tenham sido executados.

Os Comandos Paralelos de Repetição

Os comandos de repetição de Actus são REPEAT, WHILE e FOR. Para os comandos REPEAT e WHILE a expressão boleana de controle pode ser escalar ou paralela. Para o comando FOR a expressão deve ser sempre escalar.

O comando WHILE tem a seguinte forma:

WHILE *boolean_expression* **DO** *statement*

Se a expressão boleana for escalar os comandos em *statement* são executados repetidamente sem alterar o escopo da **edp** vigente, e portando sob sua influência. Estes comandos são executados sob influência daquela **edp** enquanto o valor de *boolean_expression* for TRUE.

Se a expressão boleana for paralela, cada vez que esta expressão for avaliada antes de cada iteração, uma nova máscara será criada, alterando o valor da **edp** ativa para aquela iteração.

O comando REPEAT tem a seguinte forma.

REPEAT *statements* **UNTIL** *boolean_expression*

Este comando é uma variação do comando WHILE e comporta-se exatamente como aquele com respeito à variação da **edp** válida dentro do comando, com a diferença de que a **edp** é alterada ao fim de cada iteração.

O comando FOR tem a seguinte forma.

FOR *var* := *expr1* **TO** *expr2* **DO** *statement*
FOR *var* := *expr1* **DOWNTO** *expr2* **DO** *statement*

Neste comando *var* e *expr* são escalares e os comandos de *statement* sempre atuam sob a **edp** vigente antes do comando FOR. A expressão *expr1* define o valor inicial de *var* e a expressão *expr2* define o valor final de *var*. A cada iteração o valor da variável *var* é incrementado ou decrementado de 1, dependendo da direção TO ou DOWNTO definida no comando. O número de iterações é calculada antes da primeira iteração e não pode ser alterado pelos comandos executados a cada iteração.

O Aninhamento dos Comandos Paralelos

Quando um comando paralelo (excluindo-se o comando de atribuição) faz parte do corpo de outro comando paralelo, se não ocorrer redefinição da **edp** por um comando USING, a **edp** vigente dentro do novo comando será a **edp** vigente antes da abertura deste novo comando, alterada pela máscara criada pela expressão de controle do comando. Quando o comando paralelo é fechado a **edp** vigente passa a ser a que estava em vigor imediatamente antes deste comando ser aberto.

Deslocamento de Índices

Para permitir uma maior versatilidade no acesso aos elementos de uma variável paralela, dois operadores de deslocamento são oferecidos: o operador SHIFT e o operador ROTATE. Estes operadores atuam sobre o conjunto de índices sem alterar o número de elementos que contem, deslocando-os para a direita ou para a esquerda em um movimento circular (ROTATE) ou unidirecional (SHIFT). Cria-se assim uma nova **edp** que permanece contudo compatível com a anterior. Os operadores de deslocamento podem ser usados em qualquer variável paralela, em ambos os lados do comando de atribuição.

3.3 Funções e Procedimentos

Actus segue as mesmas regras de Pascal quanto às declarações de funções e procedimentos, bem como de seus parâmetros. O mesmo ocorre para o modo como são chamados nos comandos.

4 As Principais Características de C_Actus

Com base nas experiências adiquiridas em [2], [4] e [3], pequenas modificações, porém significativas, foram feitas na linguagem Actus. A linguagem Oberon [19] também derivada de Pascal, embora orientada para outra classe de aplicação serviu de modelo para as características que foram introduzidas em Actus, para facilitar a programação e o processo de compilação. Também foi incluída a facilidade do programador utilizar funções disponíveis para a linguagem **C** e introduzir linhas de comando escritas nessa linguagem.

Passamos a denominar esta linguagem de C_Actus, onde ao contrário de Pascal, as letras maiúsculas e minúsculas passam a ser diferenciadas, os índices dos *arrays* iniciam sempre em zero e são sempre do tipo INTEGER, os tipos sub-range e enumerados de Actus foram eliminados e os comandos RETURN, EXIT e o conceito de módulos introduzidos.

A alteração mais importante foi a dos índices dos *arrays*. Anteriormente os ajustes feitos pelo compilador para alterar a **edp** quando indexando *arrays* cujos índices não iniciavam com o mesmo valor implicava em uma grande quantidade de cálculo, diminuindo a eficiência do código gerado para as expressões paralelas. As considerações feitas por Wirth em [19] justificando esta opção são aqui ainda mais consistentes.

A sintaxe para declaração dos *arrays* é:

ArrayType = **ARRAY** *length* { "." *length* } **OF** *type*
length = *ConstExpression*

Outra alteração importante, considerando a clareza dos programas, é a incorporação de módulos. Um módulo é uma coleção de declarações de constantes, tipos, variáveis e procedimentos, juntos com uma seqüência de comandos a serem executados no início da execução do programa. Cada módulo forma uma unidade de compilaçao, possuindo uma lista de

módulos dos quais é cliente. Os identificadores que podem ser exportados (visíveis fora do bloco) devem ser marcados como tal.

4.1 A Linguagem Híbrida C_Actus

Uma característica marcante de C_Actus é a sua ligação com a linguagem C. Além dos programas serem traduzidos para C, é permitido o uso em C_Actus das funções da biblioteca de C e da inserção de comandos diretamente em C. Os identificadores utilizados no programa em C_Actus são mantidos, para permitir o uso de depuradores de C. Contudo devido a existência de algumas diferenças nas regras de escopo das duas linguagens alguns identificadores são renomeados.

A opção de traduzir o programa fonte para a linguagem C evitará que Actus fique limitada a uma única máquina, como em [2] onde a linguagem OCCAM foi adotada como linguagem intermediária, ou o trabalho de escrever várias versões do compilador, uma para cada máquina alvo.

Escolheu-se a linguagem C (padrão ANSI) por ser (junto com Fortran-77) uma das linguagens mais utilizadas em supercomputadores [20]. C contem uma grande variedade de tipos de dados, bom conjunto de comandos de controle de fluxo do programa, pré-processador padrão, várias opções de alocação de memória, comunicação com o ambiente de programação, extensa biblioteca e acesso às características sobre ponto-flutuante da implementação.

Muitas das construções seqüênciais de C_Actus são bastantes semelhantes às de C, facilitando a ação do compilador e permitindo que o operador possa analisar o código objeto se necessário durante a depuração do programa.

Considerando-se a programação numérica e científica e comparando-se C com Fortran-77 [20] têm-se como deficiências de C :

1. Ponteiros que limitam a ação dos otimizadores.
2. Loop "for" não pode ser analisado estaticamente, a contagem do número de iterações do loop pode mudar em tempo de execução.
3. Ausência de números complexos.
4. Variável global informando erro na operação aritimética inibe vetorização ou paralelização do loop.

As deficiências apontadas no item 1 podem ser contornadas pelo compilador C_Actus impondo algumas restrições ao uso de ponteiros, permitindo que otimizadores assumam que ponteiros sempre apontam para conjuntos disjuntos. Como o comando FOR em C_Actus não permite a alteração na variável de controle, a deficiência apontada em 2 não existe.

As deficiências apontadas nos itens 3 e 4 não podem ser contornadas utilizando-se um compilador C padrão. Contudo, todas as deficiências conhecidas de C estão sendo estudadas pelo comitê X3J11.1 (Numerical C Extensions Group) da ANSI, e acréscimos estão sendo considerados para eliminar tais deficiências, aumentando o suporte de C para programação numérica e científica, além de manter a compatibilidade com o padrão atual.

4.2 Modularidade em C_Actus

Um programa em C_Actus pode ser decomposto em vários módulos, podendo cada módulo ser compilado separadamente. Um módulo deve estar contido em um único arquivo (ver figura 1). Os arquivos que constituem esses módulos devem ter desinência `cac`.

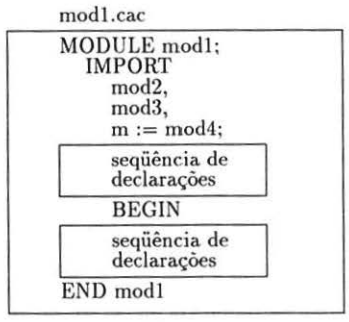

Figura 1: Estrutura de um módulo em C_Actus.

Qualquer identificador contido na seqüência de declarações e não sendo local a nenhum procedimento pode ser exportado. Os identificadores exportados são marcados na declaração com um * se disponíveis para leitura e escrita e com um - se disponíveis apenas para leitura. Os demais identificadores não são visíveis externamente. É permitido declarar pseudônimos para os módulos importados.

No exemplo a seguir as variáveis `b`, `c` e `d` são importadas dos módulos `mod2`, `mod3` e `mod4`. Este através de seu pseudônimo.

```
a := mod2.b + mod3.c - m.d;
```

Durante o processo de compilação é extraído um arquivo `*.def` contendo apenas os identificadores exportáveis, com o respectivos tipos e classe de leitura/escrita. O arquivo C terá o mesmo nome do módulo com a desinência alterada para `c`. Para o exemplo acima o módulo `mod1.cac` após a compilação irá gerar os arquivos `mod1.c` e `mod1.def`. O módulo `mod1.c` terá a seguinte formação, onde as variáveis globais não exportáveis são declaradas como `static`.

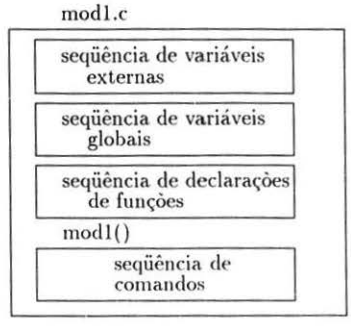

Figura 2: Estrutura do módulo traduzido para C.

Quando um módulo aparece na lista de IMPORT, todas as informações necessárias sobre o mesmo poderão ser buscadas no arquivo `*.def` correspondente. Para os módulos que exportam procedimentos e tipos de dados será também criado um arquivo `*.h` contendo suas definições para uso do compilador C.

4.3 O Processo de Compilação

O programa fonte em C_Actus é processado pelo compilador C_Actus [6], gerando um programa fonte em C. O programador deve informar ao compilador o tipo de máquina alvo. Para uma depuração preliminar da sua correção lógica, desde que não esteja utilizando nenhuma função exclusiva da biblioteca de C paralelo, o programa pode ser compilado para rodar em uma máquina seqüencial (por exemplo uma estação Sun). Posteriormente, após a depuração do modelo seqüencial, o programa tornará a ser compilado, gerando código para uma das máquinas paralelas suportada pelo compilador.

O programa objeto deve então ser compilado ou pelo compilador C (padrão UNIX ou ANSI para máquina seqüencial) ou C paralelo para as máquinas NCPI8 e iPSC 860.

No código objeto em C, tanto os identificadores como a estrutura do programa mantém uma certa semelhança com o programa fonte (em C_Actus). Isto é uma vantagem durante o processo de depuração do programa, onde eventuais mensagens de erro geradas pelo compilador C poderão ser analisadas pelo programador.

Diferenças nas Regras de Escopo

Em C_Actus as variáveis importadas são qualificadas pelo nome do módulo a que pertencem, permitindo que identificadores homônimos de módulos distintos possam ser importados por um mesmo módulo. Por exemplo `mod1.val` e `mod2.val`.

Como em C os identificadores são importados sem menção ao módulo (arquivo) a que pertencem, todos os identificadores em C_Actus marcados como visíveis externamente ao módulo, para evitar conflitos, recebem outro nome em C. O novo nome é formado pela concatenação do caracter "_" com o nome do módulo e o nome do identificador.

Outra possibilidade de conflito ocorre com o nome dos procedimentos. Como em C todos os procedimentos (funções no jargão de C) são visíveis dentro do módulo a que pertencem, procedimentos homônimos em C_Actus (contidas em blocos diferentes) devem também ser renomeados.

Um problema inverso ocorre com os identificadores que são visíveis por procedimentos declarados dentro de outros procedimentos. Em C todos os identificadores declarados dentro de uma função não são visíveis fora dela. Para contornar esse problema, na tradução para C os identificadores internos a um procedimento `A` utilizados por uma procedimento `B`, são acrescentados à lista dos parâmetros de `B`.

Palavras reservadas

Além das palavras reservadas de Actus, o programador não deve utilizar as palavras reservadas de C. Seria possível alterar o nome de todos os identificadores que utilizassem uma destas palavras aumentando o grau de liberdade do programador ao nomear seus identificadores mas aumentando a complexidade do compilador.

5 A Implementação de C_Actus

As construções paralelas de C_Actus além de aumentar a clareza do programa, e fornecer uma notação mais compacta na qual o programador possa escrever seus algoritmos, devem permitir que o compilador possa gerar código eficiente explorando ao máximo os recursos da máquina alvo.

As máquinas alvo inicialmente escolhidas para C_Actus não estão na mesma classe para as quais Actus foi projetada (*Array e Vector processors*). Ainda está em aberto a questão

[6] O compilador também é escrito em C.

de como serão tratadas pelo compilador as construções paralelas, se serão necessárias mais alterações na linguagem, e qual o nível interferência a ser feita pelos otimizadores.

Para permitir que C_Actus possa ser utilizado em várias áreas de pesquisas e que alterações e acréscimos à linguagem possam ser implementados e avaliados mais rapidamente, esforços têm sido feitos para fornecer um bom nível de documentação do compilador.

A árvore sintática construída durante o processo de compilação, e que eventualmente será alterada pelos otimizadores pode ser examinada na sua totalidade ou em pontos específicos, colocando-se diretivas no texto do programa fonte, numa tentativa de facilitar a implementação e depuração dos otimizadores. Isto é ainda mais significativo porque várias técnicas compátiveis ou não, devem ser avaliadas.

Inicialmente será oferecida uma linguagem cuja notação paralela é traduzida em um programa seqüencial, que pode eventualmente sofrer algum tipo de otimização pelos compiladores **C**. Por permitir acesso às funções de **C**, um programa em C_Actus, em um ambiente que disponha de um compilador **C** paralelo, pode incorporar características, tais como: alocação de processadores, distribuição de dados e troca de mensagens entre processos, que o tornam adeqüado a ambientes de programação MIMD, além do SIMD para o qual foi originalmente concebida.

Como em Actus, C_Actus limita a dois o número de dimensões que podem ser executadas em paralelo. Contudo este valor pode ser aumentado se durante a fase de avaliação e teste constatar-se que dimensões maiores se fazem necessárias e podem ser eficientemente implementadas.

Os programas C_Actus que não contenham chamadas de funções exclusivas à biblioteca de **C** paralelo podem ser executados em máquinas seqüênciais, como em uma estação SUN, com grandes recursos para auxílio na depuração de programas. A seguir apresentamos sucintamente como são traduzidas algumas das contruções paralelas de C_Actus. Como o resultado alcançado em uma execução seqüencial é o correto, todas as versões paralelas do programa devem alcançar o mesmo resultado, fornecendo assim um bom ambiente para implementação e teste das técnicas de paralelização de código.

O Comando USING

O código seqüencial em **C** é naturalmente mais extenso, perdendo muito da simplicidade original. Mostramos abaixo exemplos de tradução de vários comandos de C_Actus para **C**. Para maior clareza, esses exemplos são apresentados na sua forma mais simples sem ter sido feita nenhuma otimização.

Um comando USING é decomposto em um comando `for`, onde a cada iteração é avaliado os elementos relativos a um dos índices. Havendo dependência de dados entre comandos, para manter a semântica correta, será necessário separar o loop em vários comandos `for`.

```
VAR A,B: ARRAY 100 OF INTEGER;    (* PROGRAMA EM C_ACTUS *)
VAR C,D: ARRAY 100 OF REAL;

USING IS1 := 8:[2]80 DO
  BEGIN
    A[IS1] := B[IS1] * 2;
    C[IS1] := D[IS1] - B[IS1];
    B[IS1] := A[IS1];
  END
```

```
for(i=8;i<80;i+=2;)                    /* programa em C */
{
  A[i] = B[i] * 2;
  C[i] = D[i] - B[i];
}
for(i=0;i<80;i+=2)
{
  B[i] = A[i];
}
```

Os Deslocamentos de Índices

As operações de SHIFT com os índices consistem em somar ou subtrair do índice o valor do deslocamento. As operações de ROTATE são um pouco mais trabalhosas pois há a necessidade de verificar se o índice ultrapassou seu limite inferior ou superior e fazer o ajuste necessário.

```
USING IS1 := 8:[2]80 DO              (* PROGRAMA EM C_ACTUS *)
  A[IS1 SHIFT 3] := B[IS SHIFT -4] * 2;
  C[IS1 ROTATE 5] := D[IS1 SHIFT 3] - B[IS1 ROTATE-7];

for(i=8;i<80;i+=2)              /* programa em C */
{
  A[i+3] = B[i-4] * 2;
  C[(i+5)%100] = D[i+3] - B[(i+93)%100];
}
```

O Comando IF

Para cada comando IF é necessário associar-se um *array* temporário para formar uma máscara que definirá que índices da **edp** serão utilizados nas cláusulas THEN e ELSE. Essa máscara terá a dimensão do comando USING ativo.

```
USING IS:=0:[3]70, JS:=5:40 DO    (* PROGRAMA EM C_ACTUS *)
  IF AA[IS,JS] >= 0.0 THEN
    BB[IS,JS] := BB[IS,JS] - 0.5;
  ELSE
    BB[IS,JS] := BB[IS,JS] + 0.5;

for(i=0;i<70;i+=3)               /* programa em C */
  for(j=5;j<40;j+=1)
    temp[i][j] = AA[i][j] >= 0.0;
for(i=0;i<70;i+=3)
  for(j=5;j<40;j+=1)
    if(temp[i][j]==TRUE)
      BB[i][j] = BB[i][j] - 0.5;
    else
      BB[i][j] = BB[i][j] + 0.5;
```

O Comando WHILE

Assim como no comando FOR, um *array* temporário deve servir de máscara para definir que índices da **edp** estarão ativos a cada iteração do comando WHILE. Note que a cada iteração o tamanho da **edp** nunca aumenta. A função **any** retorna TRUE se algum elemento do *array* for TRUE.

```
WHILE AA[IS,JS] >= 0 DO          (* PROGRAMA EM C_ACTUS *)
  BEGIN
    BB[IS,JS] := BB[IS,JS] + BB[IS,JS] * AA[IS,JS];
    AA[IS,JS] := AA[IS,JS] - 4;
  END
```

```
for(i=0;i<70;i+=1)  /* programa em C */
  for(j=0;j<40;j+=1)
    temp[i][j] = FALSE;
for(i=0;i<70;i+=3)
  for(j=5;j<40;j+=1)
    temp[i][j] = AA[i][j] >= 0;
while(any(temp,69,39))
{
  for(i=0;i<70;i+=3)
    for(j=5;j<40;j+=1)
    {
       BB[i][j] = BB[i][j] + BB[i][j] * AA[i][j];
       AA[i][j] = AA[i][j] - 4;
    }
  for(i=0;i<70;i+=3)
    for(j=5;j<40;j+=1)
      if(temp[i][j] == TRUE )
        temp[i][j] = AA[i][j] >= 0;
}
```

6 Conclusão

O projeto de C_Actus consiste no desenvolvimento conjugado de uma linguagem de programação paralela, do seu compilador, e do estudo de técnicas de otimização de código. Programas reais (atualmente escritos em FORTRAN) serão escritos em C_Actus para verificar a adeqüação e desempenho da linguagem. As características da linguagem poderão ser reavaliadas durante os trabalhos e alterações feitas tanto na linguagem como no compilador.

Dois outros pontos importantes são a portabilidade entre máquinas e a facilidade de se testar programas com paralelismo vetorial executando-se a sua versão seqüencial.

Por gerar código em **C** e permitir que comandos de **C** possam ser enxertados nos programas em C_Actus, todas as funções disponíveis para **C** podem ser acessadas. Assim testes de desempenho poderão ser feitos nas várias máquinas disponíveis, utilizando-se a versão preliminar do compilador permitindo uma maior agilidade na sua avaliação e alteração.

Ao paralelismo SIMD de C_Actus é acrescido o paralelismo MIMD de **C** paralelo, oferecendo aos otimizadores uma maior gama de opções e acesso aos recursos das máquinas alvo.

Procura-se facilitar e incentivar o acesso de mais pesquisadores ao projeto através de uma boa documentação do compilador e ferramentas que permitam visualizar e alterar sua representação intermediária.

Uma versão operacional do compilador está sendo concluída e testada.

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 Distrituted Computing*, 4 (1987), 266-287.

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

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

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

[5] Amorim, C.L., Citro, R., Souza, A.F., Chaves Filho, E.M., "The NCPI Parallel Computer System", *Thecnical Report ES-241/1991, April.*

[6] "$i860^T M$ 64-bit Microprocessor Hardware Reference Manual", Intel Corporation.

[7] B.W. Kernighan and D.M. Ritchie, "The C Programming Language", (2nd ed) Prentice Hall, 1988.

[8] "Helios C Manual", Periehelion Software Limited, 1989.

[9] Carneiro, M.C.R., "Implementação de um Compilador para a Parte Seqüencial da Linguagem Actus/Oberon", *Projeto de Fim de Curso, Dpto de Informática, IM/UFRJ*, Maio de 1993.

[10] Polychronopoulos, C.D., "Parallel Programming and Compilers", Kluwer Academic Publishers, 1988.

[11] Wolfe, M.J., "Data Dependence and Program Restructuring", *The Journal of Supercomputing*, vol 4, pags. 321-344, Jan. 1991.

[12] Wholey, S., "Automatic Data Mapping for Distributed-Memory Parallel Computers", *ACM Trans. Programming Languages Syst.* pags. 25-34, 1992.

[13] Hiranandani, S., Kennedy, K., Tseng, C., "Compiling Fortran D For MIMD Distributed Memory Machines", *Communications of the ACM* vol 35, ago 1992.

[14] Balasundaram, V., Fox, G., Kennedy, K., Kremer, U., "An Interactive Environment for Data Partitioning and Distribution", *Fifth Distributed Memory Computing Conference*, Charleston, S. Carolina, April 9-12, 1990.

[15] Callahan, D., Kennedy, K., "Compiling Programs for Distributed Memory Multiprocessors", *Journal of Supercomputing*, October, 1988.

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

[17] "High Perfomance Fortran Language Specification", Jan. 25, 1993, Version 1.0 DRAFT.

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

[19] Wirth, N., "From Modula to Oberon", *Software – Practice and Experience*, Vol. 18 (1988), 661-670.

[20] MacDonald, T. "C Versus Fortran-77 for Scientific Programming", *Scientific Programming*, Vol. 1 (1992), 99-114.