

Paralelização Automática de Código em Plataformas Heterogêneas Modernas

Rogério A. Gonçalves^{1,2} Alfredo Goldman²

¹Departamento de Computação (DACOM)
Universidade Tecnológica Federal do Paraná (UTFPR)
Grupo de Computação de Alto Desempenho e Sistemas Distribuídos

²Laboratório de Sistemas de Software
Instituto de Matemática e Estatística (IME)
Universidade de São Paulo (USP)

rogerioag@utfpr.edu.br, {rag,gold}@ime.usp.br

MAC5742 - Computação Paralela e Distribuída



Roteiro

- 1 Introdução
- 2 Conceitos
- 3 Abordagens sobre Paralelização
- 4 Transformações de Laços
- 5 Modelo Polyhedral
- 6 Ferramentas e Exemplos
- 7 LLVM e Ferramentas
- 8 Proposta
- 9 Dúvidas

Objetivos

- Uma visão geral sobre a abordagem de paralelização automática.
- Apresentar as ferramentas e projetos LLVM.
- Alguns conceitos sobre o Modelo *Polyhedral*.
- Apresentar algumas ferramentas que utilizam esse modelo.
- Exemplos práticos indo do fonte original ao código para GPU.

- O hardware tem evoluído rapidamente e as plataformas de processamento paralelo tornam-se cada vez mais heterogêneas.
- Dispositivos aceleradores, como GPUs e arranjos de coprocessadores tem sido utilizados na composição do hardware de supercomputadores.
- Nessas novas plataformas, não há suporte direto para as aplicações existentes serem reescritas.
- Os *kits* de desenvolvimento ainda não estão maduros o suficiente.
- Programar novas aplicações ou traduzir aplicações de código legado para estas novas plataformas não é uma tarefa trivial.

- O processo de reescrever o código é oneroso e em muitas situações pode ser impraticável.
- Existem diversas tentativas que buscam facilitar o desenvolvimento e atrair usuários de outros contextos.
- *Bindings* e bibliotecas tem sido desenvolvidas para facilitar o acesso a outras linguagens.
- Ponto de vista do software.

Plataformas vs. Aplicações

No cenário atual é evidente a grande distância entre o potencial de desempenho disponibilizado pelas plataformas modernas e as aplicações que precisam fazer uso desse potencial.

- São necessárias soluções que deem suporte a execução de código de aplicações novas e de código legado.
- O ideal é que a ligação entre esses contextos seja feita por ferramentas.

- Existem pelo menos duas formas de se obter aplicações paralelas:
 - Concepção de aplicações paralelas: escrever código em uma linguagem de programação com suporte a paralelismo (aplicações naturalmente paralelas).
 - Adaptar aplicações existentes: reescrever código usando extensões que deem o suporte às versões paralelas dessas aplicações (adaptação de modelos).
- Nas duas formas, existe um modelo de programação que fornece suporte à escrita ou adaptação do código, e um modelo de execução que irá executar a versão paralela do código.

Paralelização de Aplicações II

- No contexto de tradução de código, tem-se pelo menos duas abordagens conhecidas de paralelização que se distinguem na forma e no momento da intervenção para paralelização:
 - *Via código fonte*: depende da existência do código fonte (*source-to-source*).
 - *Via código binário*: aplica transformações sobre o binário gerado para a execução da aplicação. (Tradução Dinâmica - DBT).

- A aceleração de aplicações está relacionada com as plataformas modernas.
- São plataformas heterogêneas, que congregam diversas tecnologias e seus elementos de processamento que trabalham como aceleradores na execução de aplicações.
- O uso de dispositivos aceleradores torna possível a divisão da carga de execução entre um elemento de processamento principal, geralmente uma CPU e elementos que irão trabalhar como coprocessadores.

- Como exemplos dos dispositivos que atualmente compõem essas plataformas temos os arranjos de coprocessadores Xeon Phi (Intel), as GPUs (NVidia, AMD) e FPGAS (Xilinx, Altera).
- Regiões de código paralelizáveis podem ser transformadas em *kernels* e terem sua execução acelerada em uma GPU. Ex.: Laços.
- Trabalhos tem proposto modelos para a execução de aplicações combinando dispositivos *multicore* e multiGPU.

Abordagens que se destacam

Diretivas de Compilação

Diretivas de compilação são usadas para guiar o compilador no processo de tradução e paralelização de código.

Paralelização Automática

São usadas técnicas e modelos para detectar automaticamente quais regiões do código são paralelizáveis.

- Nas duas abordagens uma versão paralela do código é gerada para a plataforma alvo.

Diretivas de Compilação

- O código é anotado com diretivas de preprocessamento (#pragma) que são tratadas e o código é gerado. Definições e macros são substituídas pelo código que fará o chaveamento entre dispositivos no processo de ligação com o ambiente de execução.

Ferramentas que utilizam diretivas

- O OpenMP.
- O hiCUDA (transformar e gerar código para CUDA).
- O CGCM (otimiza a comunicação e transferências de dados).
- Os compiladores PGI e o OpenHMPP que implementam o padrão OpenACC.
- O accULL é uma implementação do padrão OpenACC com suporte a CUDA e a OpenCL.
- O OpenMC é uma extensão do OpenMP para suporte a GPU.

Diretivas de Compilação

- Exemplo da diretiva `#pragma acc kernels` que define uma região de código que será transformada em *kernels*.

```
1 #pragma acc data copy(b[0:n*m]) create(a[0:n*m])
2 {
3     for (iter = 1; iter <= p; ++iter) {
4         #pragma acc kernels
5         {
6             for (i = 1; i < n-1; ++i)
7                 for (j = 1; j < m-1; ++j) {
8                     /* Suprimido. */
9                 }
10            for( i = 1; i < n-1; ++i )
11                for( j = 1; j < m-1; ++j )
12                    /* Suprimido. */
13            }
14        }
15 }
```

Paralelização Automática

- A ideia é que o código não seja modificado (nem anotado), pois a ferramenta deve detectar automaticamente as regiões paralelizáveis.

Ferramentas da abordagem automática

- Par4All (análise interprocedural)
- C-to-CUDA
- PPCG
- KernelGen

Transformações de Laços I

- A análise de dependência para laços ser feita sobre todas as iterações do laço, como se o laço fosse todo desenrolado.
- Pode acontecer de toda iteração do laço usar valores calculados pela iteração anterior (exceto a primeira).
- Técnicas de otimização de laços são utilizadas por compiladores como mecanismo para a melhoria de desempenho.

Transformações de Laços II

- Essas transformações são muito importantes pois possibilitam a exploração de capacidades do processamento paralelo e ajudam a reduzir o tempo de execução associado aos laços.
- O processo de otimização de laços pode ser definido como uma sequência de transformações aplicadas aos laços de um código ou à sua representação intermediária.
- Uma transformação aplicada deve preservar a semântica do código original e a sequência temporal de todas as dependências.

Transformações de Laços III

- Existem diversas técnicas específicas para otimização de laços.
- São classificadas conforme as modificações que produzem no código e de acordo com o efeito que garantem na execução do código final.
- Transformações podem ser aplicadas para: analisar, preparar e expor algum possível paralelismo latente.
- São exemplos de otimizações que reorganizam o código de laços: *loop unrolling*, *loop interchange*, *loop skewing*, *loop reversal*, *loop blocking* e *cycle shrinking*, *loop fusion*, *peeling* e distribuição, entre outras.

Transformações de Laços: *Loop Tiling* I

- É uma transformação que reorganiza o laço para iterar sobre blocos (*blocking*) de dados.
- O algoritmo clássico de multiplicação de matrizes é um exemplo ao qual esse transformação pode ser aplicada.

```
1 for( int i=0; i<N; i++) {  
2     for( int j=0; j<M; j++) {  
3         C[ i ][ j ] = 0;  
4         for( int k=0; k<N; k++) {  
5             C[ i ][ j ] = C[ i ][ j ] + A[ i ][ k ] * B[ k ][ j ];  
6         }  
7     }  
8 }
```

Transformações de Laços: *Loop Tiling* II

- O código pode ser transformado para iterar sobre blocos (*blocking*).

```
1 /* B: blocking factor ou tile size. */
2 for (int jj=0; jj<N; jj=jj+B) {
3     for (kk=0; kk<N; kk=kk+B) {
4         for (i=0; i<N; i++) {
5             for (j=jj; j<min(jj+B-1, N); j++) {
6                 C[i][j] = 0;
7                 for(k=kk; k<min(kk+B-1, N); k++) {
8                     C[i][j] = C[i][j] + A[i][k] * B[k][j];
9                 }
10            }
11        }
12    }
13 }
```



Transformações de Laços: *Loop Skewing* |

- É uma transformação que reorganiza o espaço de iteração de laços aninhados que iteram sobre arranjos multidimensionais, onde cada iteração do laço mais interno depende de iterações anteriores.
- Os limites do espaço de iteração são alterados para que iterações possam ser executadas em paralelo.
- Tomemos como exemplo o código com dois laços aninhados que fazem acesso a um arranjo multidimensional calculando cada elemento com base nos elementos vizinhos (wavefront).

```
1 for (i=1; i<=n-1; i++) {  
2     for (j=1; j<=m-1; j++) {  
3         A[i][j] = (A[i-1][j] + A[i][j-1] + A[i+1][j] + A[i][j+1]) / 4;  
4     }  
5 }
```

Transformações de Laços: *Loop Skewing* II

- O espaço de iteração foi deslocado para que iterações possam ser executadas em paralelo.

```
1 if ((m >= 2) && (n >= 2)) {  
2     for (c1=2; c1<=n+m-2; c1++) {  
3         for (c2=max(1,c1-m+1); c2<=min(c1-1,n-1); c2++) {  
4             A[c2][c1-c2] = (A[c2-1][c1-c2] + A[c2][c1-c2-1] + A[  
5                 c2+1][c1-c2] + A[c2][c1-c2+1]) / 4;  
6         }  
7     }
```

- Este exemplo será utilizado para mostrar a ideia do Modelo *Polyhedral*.

Transformações de Laços e Modelo Polyhedral I

- O Modelo *Polyhedral* possui um suporte melhor para a representação e análise de código de laços do que a AST.
- A representação de SCoPs no modelo baseia-se nos conceitos: domínio de iteração, função de *scattering* e função de acesso.
- Encontrando-se um SCoP, a região de código pode ser representada no modelo e transformações e otimizações podem ser aplicadas à representação gerada.

```
1 for ( i=1; i<=n-1; i++) {  
2     for ( j=1; j<=m-1; j++) {  
3         A[i][j] = (A[i-1][j] + A[i][j-1] + A[i+1][j] + A[i][j+1])  
            /4;    // S1  
4     }  
5 }
```

Transformações de Laços e Modelo Polyhedral II

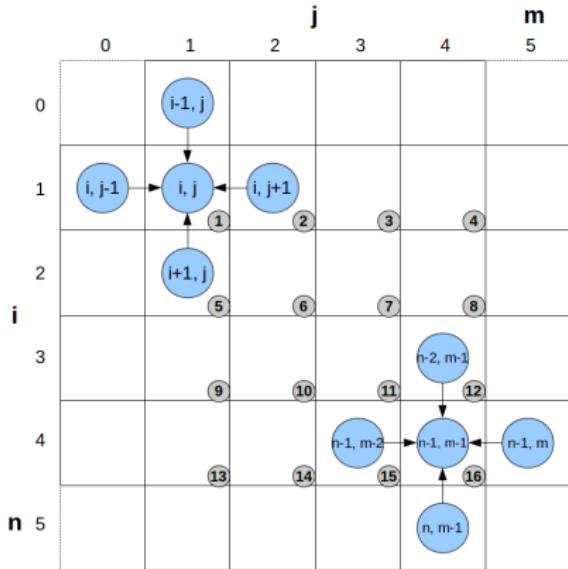


Figura 1: Esquema de acesso e cálculo dos elementos

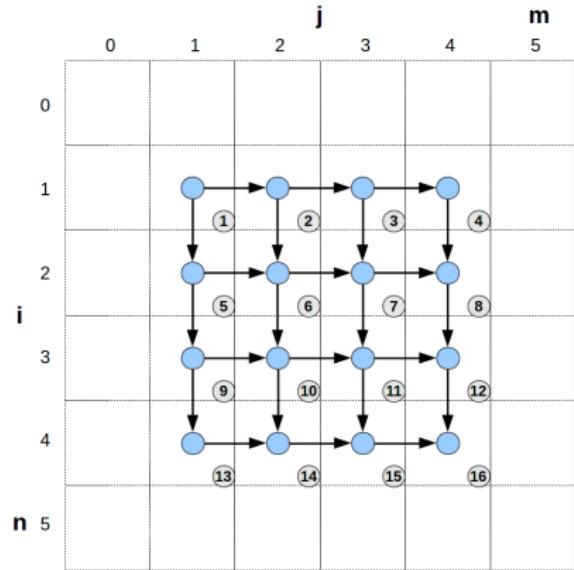


Figura 2: Dependências no uso dos elementos

Transformações de Laços e Modelo Polyhedral III

Formalmente o Domínio de Iteração para S1 pode ser descrito como:

$$D_{S1} = \{(i, j) \in \mathbb{Z}^2 \mid 1 \leq i \leq n - 1 \wedge 1 \leq j \leq m - 1\} \quad (1)$$

E o conjunto de restrições para S1(i, j):

$$\begin{array}{ll} 1 \leq i & 1 \leq j \\ 1 \leq i \leq n - 1 & 1 \leq j \leq m - 1 \\ i \leq n - 1 & j \leq m - 1 \\ i - 1 \geq 0 & j - 1 \geq 0 \\ n - i - 1 \geq 0 & m - 1 - j \geq 0 \end{array}$$

Transformações de Laços e Modelo Polyhedral IV

Função de *Scattering*: $c_1 = i + j$ e $c_2 = i$ (novo domínio de iterações).

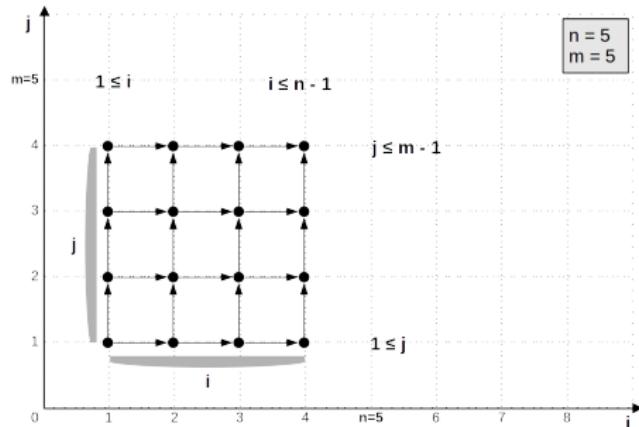


Figura 3: Domínio de Iteração

$$D_{S1} = \{(i, j) \in \mathbb{Z}^2 \mid 1 \leq i \leq n-1 \wedge 1 \leq j \leq m-1\}$$

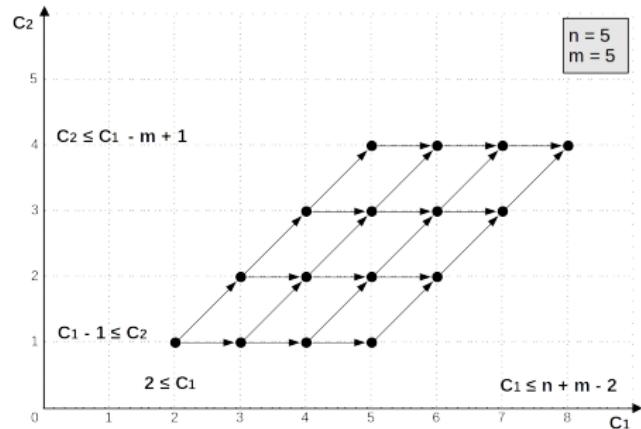


Figura 4: Função de *Scattering*

$$\theta_{S1}(i, j) = (i + j, i)$$

$$\theta_{S1}(i, j) = (c_1, c_2)$$

$$c_1 = i + j$$

$$c_2 = i$$

Transformações de Laços e Modelo Polyhedral V

- Para manipular o conjunto de restrições é utilizada uma representação matricial
- Para isso é usado um vetor de iteração com algumas dimensões adicionais para representar os parâmetros e a constante (vetor de iteração homogêneo).
- Para S1 o vetor de iteração em coordenadas homogêneas é $(i, j, n, m, 1)$.
- Então podemos escrever todas as restrições como desigualdades afins da forma *affine constraint* ≥ 0 .

Transformações de Laços e Modelo Polyhedral VI

As restrições colocadas no formato do *vetor de iteração* ($i, j, n, m, 1$, 1).

i	j	n	m	1	
i				-1	≥ 0
-i		n		-1	≥ 0
	j			-1	≥ 0
	-j		m	-1	≥ 0

Ficando portanto o sistema de restrições da seguinte forma:

$$\begin{cases} i - 1 \geq 0 \\ -i + n - 1 \geq 0 \\ j - 1 \geq 0 \\ -j + m - 1 \geq 0 \end{cases}$$

Transformações de Laços e Modelo Polyhedral VII

- Utilizando a notação matricial para
matriz de domínio \times *vetor de iteração* ≥ 0
- Essas informações do domínio de iterações (contexto) podem ser escritas como:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & -1 & 0 & 1 & -1 \end{bmatrix} \times \begin{pmatrix} i \\ j \\ n \\ m \\ 1 \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

- Essa matriz de domínio será usada para fornecer informações sobre o *domínio de iteração* de S1.
- No domínio de iteração não há informações sobre ordenação, é descrito somente o conjunto de instâncias dos *statements*, mas não a ordem na qual eles tem que ser executados.

Transformações de Laços e Modelo Polyhedral VIII

- Entretanto algumas instâncias podem depender de outras.
- Assim, é fundamental estabelecer uma ordem de execução para preservar a semântica do código original.
- Uma função de *scattering* adiciona ao modelo informações sobre a distribuição ou ordenação das instâncias.
- Existem muitos tipos de ordenação, tais como: alocação, *scheduling* (agendamento), *chunking* (conflito, colisão), que podem ser expressados da mesma maneira, porém cada uma com seu significado próprio de ordenação.

Transformações de Laços e Modelo Polyhedral IX

- Neste exemplo de *Loop Skewing* o *scattering* é feito pela função de *scheduling*:

$$\theta_{S1}(i, j) = (i + j, i) \quad (2)$$

$$\theta_{S1}(i, j) = (c_1, c_2) \quad (3)$$

$$c_1 = i + j \quad (4)$$

$$c_2 = i \quad (5)$$

- A função de *scheduling* para S1 deve ser representada também na forma de matriz:
 $\theta_S(\text{vetor de iteracao}) = \text{matriz de scattering} * \text{vetor de iteracao}.$

Transformações de Laços e Modelo Polyhedral X

- Assim, a representação de $c1$ e $c2$ das equações 4 e 5 é escrita como:

$$\theta_{S1} \begin{pmatrix} c1 \\ c2 \\ i \\ j \\ n \\ m \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & -1 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{pmatrix} c1 \\ c2 \\ i \\ j \\ n \\ m \\ 1 \end{pmatrix}$$

Transformações de Laços e Modelo Polyhedral XI

Código original que foi modificado utilizando o CLooG (*Skewing*):

```
1 for (i=1; i<=n-1; i++) {  
2     for (j=1; j<=m-1; j++) {  
3         A[i][j] = (A[i-1][j] + A[i][j-1] + A[i+1][j] + A[i][j+1])/4; // S1  
4     }  
5 }
```

Substituindo as variáveis: $i = c2$ e $j = c1 - i = c1 - c2$, tem-se o código final:

```
1 if ((m >= 2) && (n >= 2)) {  
2     for (c1=2;c1<=n+m-2;c1++) {  
3         for (c2=max(1,c1-m+1);c2<=min(c1-1,n-1);c2++) {  
4             A[c2][c1-c2] = (A[c2-1][c1-c2] + A[c2][c1-c2-1] + A[  
                c2+1][c1-c2] + A[c2][c1-c2+1])/4; //S1'  
5         }  
6     }  
7 }
```

Transformações de Laços e Modelo Polyhedral XII

- As matrizes com as definições das restrições do domínio de iteração e da função de *scheduling* podem ser colocadas manualmente no formato de entrada do CLooG.
- Ou podem ser geradas por ferramentas como o `clan` e o `pet` que detectam os SCoPs.
- Possibilitando a transformação de *loop skewing* descrita pela função de *scattering*.

Ferramentas que utilizam o Modelo Polyhedral I

- São ferramentas que cobrem desde a extração de regiões candidatas à paralelização, passando por bibliotecas que permitem o uso como subprojetos até geradores de código com base em informações do Modelo *Polyhedral*.
- Ferramentas que extraem regiões candidatas à paralelização: *clan* e *pet*. Extraem *Static Control Parts (SCoPs)*.
- O *CLooG* é um gerador de código para C.
- Compiladores como *PLuTo* e *PoCC* fazem tradução *source-to-source* de código C e Fortran.
- O *PLuTo* faz paralelização automática para *multicore* e GPU.

Ferramentas que utilizam o Modelo Polyhedral II

- O PoCC agrupa várias ferramentas, extrai *SCoPs* com o clan, analisa dependências com o candl, usa o algoritmo de escalonamento, alocação e *tiling* do PLuTo e gera código com o CLooG.
- O C-to-CUDA é uma ferramenta que faz a tradução *source-to-source* de código C para CUDA, usa a técnica de *tiling* do PLuTo com transformações para GPU e o gerador de código CLooG.
- PPCG usa as bibliotecas pet e isl para gerar código OpenMP e CUDA.
- O PoLLy é um projeto LLVM que implementa o modelo *polyhedral* para código intermediário (LLVM-IR).

Detecção de Regiões Paralelizáveis I

- No processo de paralelização de código é necessário detectar as regiões paralelizáveis.
- Essas regiões são chamadas de *Static Control Parts (SCoPs)*.
- São partes do código de uma função que podem ser vistas como uma estrutura de fluxo de controle que contem apenas laços *FOR* e *IFs*.

Detecção de SCoPs com clan e pet |

- Entre as ferramentas que extraem regiões candidatas à paralelização estão o clan e pet.
- Extraem (SCoPs) de regiões anotadas com `#pragma scop` e `#pragma endscop`.
- O clan tem opção de detectar automaticamente SCoPs (`-autoscop`) e de inserir as anotações no código (`-autopragma`) para identificar as regiões detectadas.
- O pet também tem a opção `-autodetect`.

```
1 #pragma scop
2   for ( i=1; i<=n-1; i++ ) {
3     for ( j=1; j<=m-1; j++ ) {
4       A[ i ][ j ] = ( A[ i -1 ][ j ] + A[ i ][ j -1 ] + A[ i +1 ][ j ] + A
          [ i ][ j +1 ]) / 4;
5     }
6   }
7 #pragma endscop
```

Detecção de SCoPs com clan e pet II

- Exemplo de saída gerada pelo clan com as informações do SCoP detectado:

```
1 [Clan] Info: parsing file #1 (wavefront-clan-autoscop-detection.c)
2 # [File generated by the OpenScop Library 0.9.0]
3 # [SCoPLib format]
4
5 SCoP
6
7 # ====== Global
8 # Language
9 C
10
11 # Context
12 0 4
13
14 # Parameters are provided
15 1
16 # Parameter names
17 n m
```

Detecção de SCoPs com clan e pet III

```
18
19 # Number of statements
20 1
21
22 # ===== Statement 1
23 # ----- 1.1 Domain
24 # Iteration domain
25 1
26 6 6
27 # e / i /   i      n      m /   1
28   1     1      0      0      0     -1      ## i - 1 >= 0
29   1    -1      0      1      0     -1      ## -i + n - 1 >= 0
30   1     0      0      1      0     -2      ## n - 2 >= 0
31   1     0      1      0      0     -1      ## j - 1 >= 0
32   1     0     -1      0      1     -1      ## -j + m - 1 >= 0
33   1     0      0      0      1     -2      ## m - 2 >= 0
34
35 # ----- 1.2
      Scattering
36 # Scattering function is provided
37 1
38 5 6
```

Detecção de SCoPs com clan e pet IV

```
39 # e / i /   i      j /   n      m /   l      1
40     0       0      0       0       0       0      ## c1 == 0
41     0       1      0       0       0       0      ## c2 == i
42     0       0      0       0       0       0      ## c3 == 0
43     0       0      1       0       0       0      ## c4 == j
44     0       0      0       0       0       0      ## c5 == 0
45
46 # ----- 1.3 Access
47 # Access informations are provided
48 1
49 # Read access informations
50 8 6
51     5       1      0       0       0      -1      ## [1] == i - 1
52     0       0      1       0       0       0      ## [2] == j
53     5       1      0       0       0       0      ## [1] == i
54     0       0      1       0       0      -1      ## [2] == j - 1
55     5       1      0       0       0       1      ## [1] == i + 1
56     0       0      1       0       0       0      ## [2] == j
57     5       1      0       0       0       0      ## [1] == i
58     0       0      1       0       0       1      ## [2] == j + 1
59 # Write access informations
60 2 6
```



Detecção de SCoPs com clan e pet V

```
61      5      1      0      0      0      0      ## [1] == i
62      0      0      1      0      0      0      ## [2] == j
63
64 # ----- 1.4 Body
65 # Statement body is provided
66 1
67 # List of original iterators
68 i j
69 # Statement body expression
70 A[i][j] = (A[i-1][j] + A[i][j-1] + A[i+1][j] + A[i][j+1])/4;
71
72 # ===== Options
73 <arrays>
74 # Number of arrays
75 5
76 # Mapping array-identifiers/array-names
77 1 i
78 2 n
79 3 j
80 4 m
81 5 A
82 </arrays>
```

Deteção de SCoPs com clan e pet VI

- Exemplo de saída gerada pelo pet com as informações do SCoP detectado:

```
1 start: 51
2 end: 203
3 indent:
4 context: '[m, n] -> { : n >= 0 and m <= 2147483647 and m >= 0
      and n <= 2147483647
5 }',
6 arrays:
7 - context: '[m, n] -> { : n >= 0 and m >= 0 }'
8   extent: '[m, n] -> { A[i0, i1] : i1 <= -1 + m and i1 >= 0 and
      i0 <= -1 + n and i0
9     >= 0 }'
10  element_type: int
11  element_size: 4
12 statements:
13 - line: 9
14   domain: '[m, n] -> { S_0[i, j] : j <= -1 + m and j >= 1 and i
      <= -1 + n and i >=
15     1 }'
```

Deteção de SCoPs com clan e pet VII

```
16     schedule: '[n, m] -> { S_0[i, j] -> [0, i, 0, j, 0] }'
17     body:
18         type: expression
19         expr:
20             type: op
21             operation: =
22             arguments:
23                 - type: access
24                     relation: '[m, n] -> { S_0[i, j] -> A[i, j] }'
25                     index: '[m, n] -> { S_0[i, j] -> A[(i), (j)] }'
26                     reference: __pet_ref_0
27                     read: 0
28                     write: 1
29                 - type: op
30                     operation: /
31                     arguments:
32                         - type: op
33                             operation: +
34                             arguments:
35                                 - type: op
36                                     operation: +
37                                     arguments:
```

Deteção de SCoPs com clan e pet VIII

```
38      - type: op
39          operation: +
40          arguments:
41              - type: access
42                  relation: '[m, n] -> { S_0[i, j] -> A[-1 + i, j
43                      ] }',
44                  index: '[m, n] -> { S_0[i, j] -> A[(-1 + i), (j
45                          )] }',
46                  reference: __pet_ref_1
47                  read: 1
48                  write: 0
49              - type: access
50                  relation: '[m, n] -> { S_0[i, j] -> A[i, -1 + j
51                      ] }',
52                  index: '[m, n] -> { S_0[i, j] -> A[(i), (-1 + j
53                          )] }',
54                  reference: __pet_ref_2
55                  read: 1
56                  write: 0
57              - type: access
58                  relation: '[m, n] -> { S_0[i, j] -> A[1 + i, j
59                      ] }'
```

Detecção de SCoPs com clan e pet IX

```
55         index: '[m, n] -> { S_0[i, j] -> A[(1 + i), (j)]  
56             }'  
56         reference: __pet_ref_3  
57         read: 1  
58         write: 0  
59     - type: access  
60         relation: '[m, n] -> { S_0[i, j] -> A[i, 1 + j] }'  
61         index: '[m, n] -> { S_0[i, j] -> A[(i), (1 + j)] }'  
62         reference: __pet_ref_4  
63         read: 1  
64         write: 0  
65     - type: int  
66         value: 4
```

Gerando Código com o CLooG I

- O CLooG é gerador de código para C conforme a representação no Modelo *Polyhedral*.
- Recebe definições do domínio de iteração, as alterações como funções de *scattering* e gera o código modificado.
- Tomando como base o exemplo apresentado para ilustrar os conceitos do Modelo *Polyhedral*.
- As definições originais e a função de *scattering* alterada para transformar o código foram colocadas no formato de entrada para o CLooG.



Gerando Código com o CLooG II

- Exemplo de arquivo com definições de entrada para o CLooG:

```
1 # ----- CONTEXTO -----
2 C      # linguagem C
3
4 # Contexto (restricoes nos parametros)
5 2 4          # 2 linhas e 4 colunas
6 # eq/in n m 1    eq/in: 1 para desigualdade >=0, 0 para
                  igualdade =0
7     1     1   0 -1    # 1*n + 0*m -1*1 >= 0, que e n >= 1
8     1     0   1 -1    # 0*n + 1*m -1*1 >= 0, que e m >= 1
9
10 1    # Definicao manual dos nomes dos parametros.
11 n m      # nomes dos parametros.
12
13 # ----- STATEMENTS -----
14 1    # Numero de statements.
15
16 1    # Primeiro statement: um dominio.
17 # Primeiro dominio.
```

Gerando Código com o CLooG III

```
18 4 6      # 4 linhas e 6 colunas.
19 # eq/in   i     j     n     m     1
20     1     1     0     0     -1    # i >= 1 -> i           -1 >= 0
21     1    -1     0     1     0     -1    # i <= n-1 -> -i +n   -1 >= 0
22     1     0     1     0     0     -1    # j >= 1 -> j           -1 >= 0
23     1     0    -1     0     1     -1    # j <= m-1 -> -j +m -1 >= 0
24 0     0     0                               # Para opcoes futuras.
25
26 0 # Definir manualmente o nome dos iterators, nao.
27
28 # ----- SCATTERING -----
29 1
30 2 8
31 # eq/in   c1   c2   i     j     n     m     1
32     0     1     0    -1    -1     0     0     0    # c1 - i - j = 0 -> c1 = i +
33                                j (skewing).
34     0     0     1    -1     0     0     0     0    # c2 - i = 0 -> c2 = i.
35 1 # Definir manualmente o nome das dimensoes do scattering.
36 c1 c2          # nomes das dimensoes.
```

Gerando Código com o CLooG IV

- O código modificado pela transformação *loop skewing* e gerado pelo CLooG:

```
1 if ((m >= 2) && (n >= 2)) {  
2     for (c1=2;c1<=n+m-2;c1++) {  
3         for (c2=max(1,c1-m+1);c2<=min(c1-1,n-1);c2++) {  
4             S1(c2,c1-c2);  
5         }  
6     }  
7 }
```

- O espaço de iteração foi modificado, conforme as definições.

Gerando Código com o CLooG V

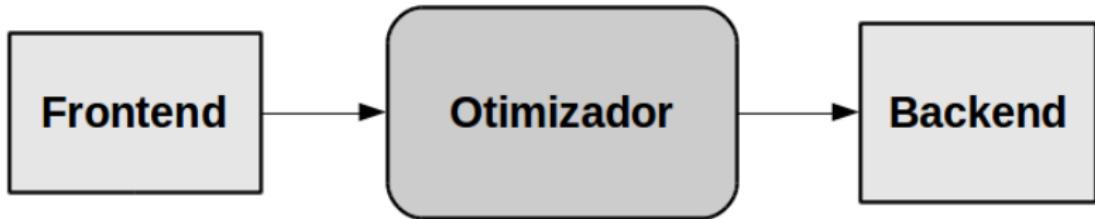
- O código gerado apresenta S1 em função das variáveis do espaço de iteração da função de *scheduling* $S1(c2, c1 - c2)$.
- Para obtermos o código final é necessário que façamos a substituição considerando as variáveis do novo espaço de iteração ($c2 = i$ e $c1 - c2 = j$).
- Fazendo as substituições na declaração original de $A[i][j]$ com os novos índices gerados para S1.

- Código Final:

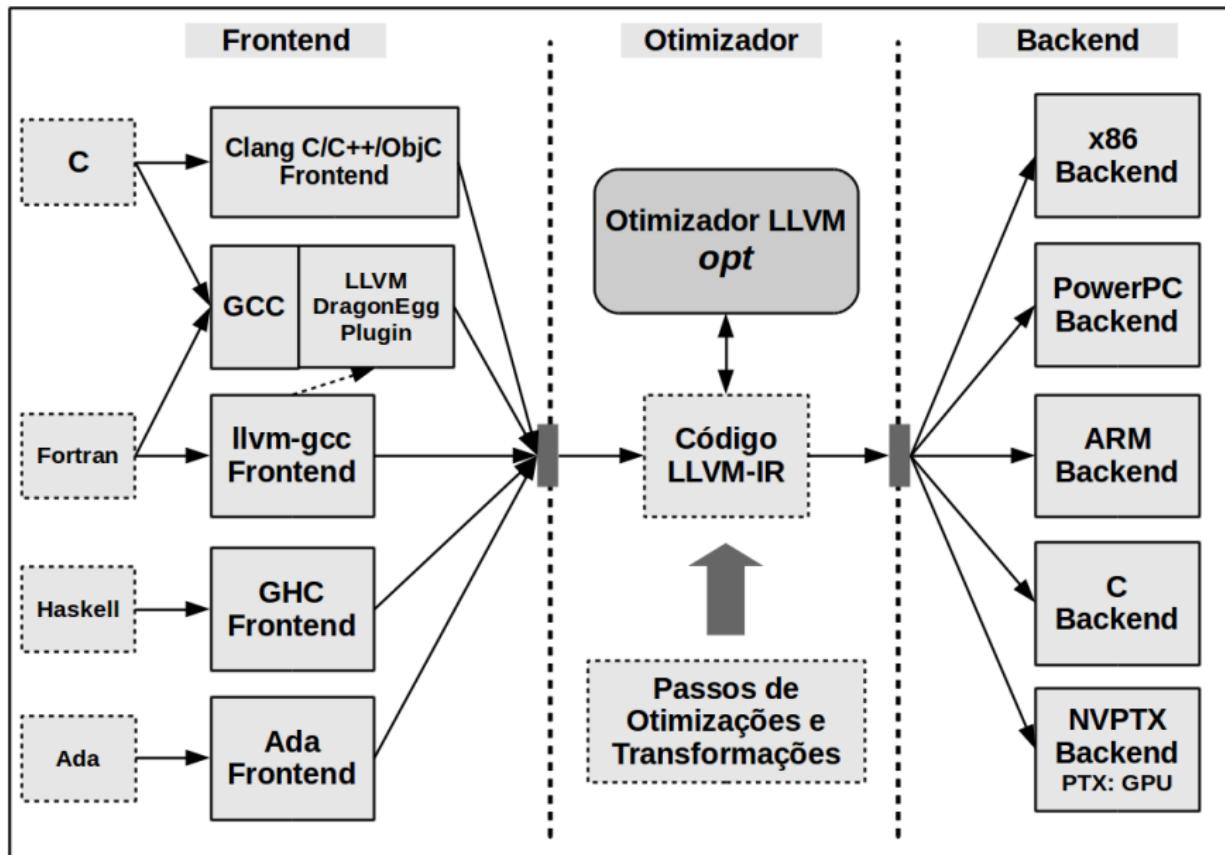
```
1 if ((m >= 2) && (n >= 2)) {  
2     for (c1=2;c1<=n+m-2;c1++) {  
3         for (c2=max(1,c1-m+1);c2<=min(c1-1,n-1);c2++) {  
4             A[c2][c1-c2] = (A[c2-1][c1-c2] + A[c2][c1-c2-1] + A[  
5                 c2+1][c1-c2] + A[c2][c1-c2+1])/4; // S1  
6         }  
7     }
```

Por que usar LLVM? I

- LLVM (llvm.org) é um projeto que fornece uma infraestrutura para a construção de compiladores.
- Desenvolvimento: LLVM Team - University of Illinois at Urbana-Champaign.
- Licença: *University of Illinois/NCSA Open Source License*, certificada pela OSI.
- Sua organização modular facilita o reuso no desenvolvimento de ferramentas de compilação.



Por que usar LLVM? II



SP

Exemplo: LLVM IR I

Código em C.

```
1 int main(){
2     int a, b, c;
3     a = 5;
4     b = 2;
5     c = a + b;
6
7     if (c > 5){
8         c = 2 * c;
9     }
10    else{
11        c = 3 * c;
12    }
13
14    return 0;
15 }
```

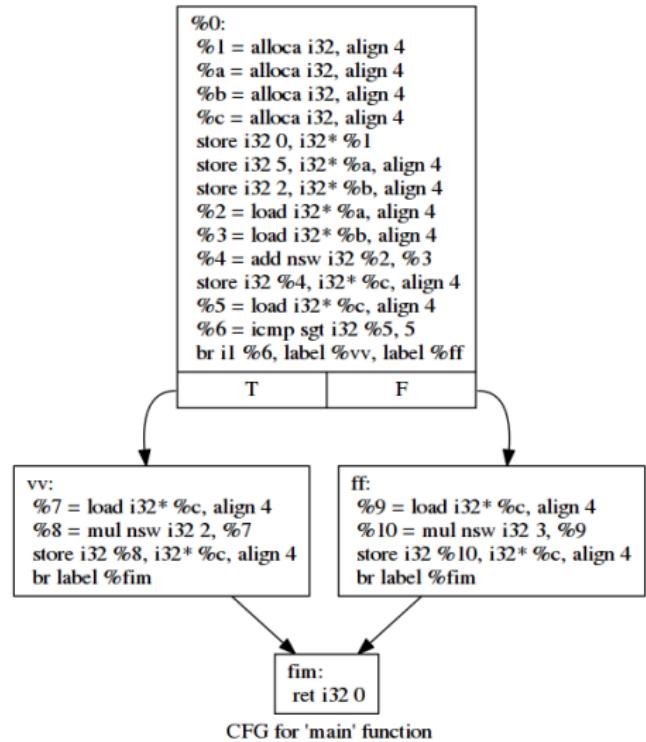
Código em LLVM-IR.

```
1 define i32 @main() #0 {
2     %1 = alloca i32, align 4
3     %a = alloca i32, align 4
4     %b = alloca i32, align 4
5     %c = alloca i32, align 4
6     store i32 0, i32* %1
7     store i32 5, i32* %a, align 4
8     store i32 2, i32* %b, align 4
9     %2 = load i32* %a, align 4
10    %3 = load i32* %b, align 4
11    %4 = add nsw i32 %2, %3
12    store i32 %4, i32* %c, align 4
13    %5 = load i32* %c, align 4
14    %6 = icmp sgt i32 %5, 5
15    br i1 %6, label %vv, label %ff
16
17 vv:                                ; preds = %0
18    %7 = load i32* %c, align 4
19    %8 = mul nsw i32 2, %7
20    store i32 %8, i32* %c, align 4
21    br label %fim
22
23 ff:                                ; preds = %0
24    %9 = load i32* %c, align 4
25    %10 = mul nsw i32 3, %9
26    store i32 %10, i32* %c, align 4
27    br label %fim
28
29 fim:                                ; preds = %vv, %ff
30    ret i32 0
31 }
```



Exemplo: LLVM IR |

```
1 define i32 @main() #0 {
2   %1 = alloca i32, align 4
3   %a = alloca i32, align 4
4   %b = alloca i32, align 4
5   %c = alloca i32, align 4
6   store i32 0, i32* %1
7   store i32 5, i32* %a, align 4
8   store i32 2, i32* %b, align 4
9   %2 = load i32* %a, align 4
10  %3 = load i32* %b, align 4
11  %4 = add nsw i32 %2, %3
12  store i32 %4, i32* %c, align 4
13  %5 = load i32* %c, align 4
14  %6 = icmp sgt i32 %5, 5
15  br i1 %6, label %vv, label %ff
16
17 vv:           ; preds = %0
18  %7 = load i32* %c, align 4
19  %8 = mul nsw i32 2, %7
20  store i32 %8, i32* %c, align 4
21  br label %fim
22
23 ff:           ; preds = %0
24  %9 = load i32* %c, align 4
25  %10 = mul nsw i32 3, %9
26  store i32 %10, i32* %c, align 4
27  br label %fim
28
29 fim:          ; preds = %vv, %ff
30  ret i32 0
31 }
```



Exemplo: LLVM IR I

Código em C.

```
1 int funcA(int x, int y){  
2     int result = x * y;  
3     return result;  
4 }  
5  
6 int main(){  
7     int a, b, c;  
8     a = 5;  
9     c = funcA(a, 2);  
10    return 0;  
11 }  
12 }
```

Código em LLVM-IR.

```
1 define i32 @funcA(i32 %x, i32 %y) #0 {  
2     %1 = alloca i32, align 4  
3     %2 = alloca i32, align 4  
4     %result = alloca i32, align 4  
5     store i32 %x, i32* %1, align 4  
6     store i32 %y, i32* %2, align 4  
7     %3 = load i32* %1, align 4  
8     %4 = load i32* %2, align 4  
9     %5 = mul nsw i32 %3, %4  
10    store i32 %5, i32* %result, align 4  
11    %6 = load i32* %result, align 4  
12    ret i32 %6  
13 }  
14 define i32 @main() #0 {  
15     %1 = alloca i32, align 4  
16     %a = alloca i32, align 4  
17     %b = alloca i32, align 4  
18     %c = alloca i32, align 4  
19     store i32 0, i32* %1  
20     store i32 5, i32* %a, align 4  
21     %2 = load i32* %a, align 4  
22     %3 = call i32 @funcA(i32 %2, i32 2)  
23     store i32 %3, i32* %c, align 4  
24     ret i32 0  
25 }
```

Exemplo: LLVM IR I

Código em C.

```
1 int main(){  
2     int i;  
3     int res = 0;  
4     for(i=0; i < 100; i++){  
5         res = res + i;  
6     }  
7  
8     return 0;  
9 }
```

Código em LLVM-IR.

```
1 define i32 @main() #0 {  
2     %1 = alloca i32, align 4  
3     %i = alloca i32, align 4  
4     %res = alloca i32, align 4  
5     store i32 0, i32* %1  
6     store i32 0, i32* %res, align 4  
7     store i32 0, i32* %i, align 4  
8     br label %2  
9  
10    ; <label>:2      ; preds = %9, %0  
11    %3 = load i32* %i, align 4  
12    %4 = icmp slt i32 %3, 100  
13    br i1 %4, label %5, label %12  
14  
15    ; <label>:5      ; preds = %2  
16    %6 = load i32* %res, align 4  
17    %7 = load i32* %i, align 4  
18    %8 = add nsw i32 %6, %7  
19    store i32 %8, i32* %res, align 4  
20    br label %9  
21  
22    ; <label>:9      ; preds = %5  
23    %10 = load i32* %i, align 4  
24    %11 = add nsw i32 %10, 1  
25    store i32 %11, i32* %i, align 4  
26    br label %2  
27  
28    ; <label>:12     ; preds = %2  
29    ret i32 0  
30 }
```



Exemplo: LLVM IR |

```
1 define i32 @main() #0 {
2   %1 = alloca i32, align 4
3   %i = alloca i32, align 4
4   %res = alloca i32, align 4
5   store i32 0, i32* %1
6   store i32 0, i32* %res, align 4
7   store i32 0, i32* %i, align 4
8   br label %2
9
10 ; <label >:2      ; preds = %9, %0
11 %3 = load i32* %i, align 4
12 %4 = icmp slt i32 %3, 100
13 br i1 %4, label %5, label %12
14
15 ; <label >:5      ; preds = %2
16 %6 = load i32* %res, align 4
17 %7 = load i32* %i, align 4
18 %8 = add nsw i32 %6, %7
19 store i32 %8, i32* %res, align 4
20 br label %9
21
22 ; <label >:9      ; preds = %5
23 %10 = load i32* %i, align 4
24 %11 = add nsw i32 %10, 1
25 store i32 %11, i32* %i, align 4
26 br label %2
27
28 ; <label >:12     ; preds = %2
29 ret i32 0
30 }
```

```
%0:
%1 = alloca i32, align 4
%i = alloca i32, align 4
%res = alloca i32, align 4
store i32 0, i32* %1
store i32 0, i32* %res, align 4
store i32 0, i32* %i, align 4
br label %2
```

```
%2:
%3 = load i32* %i, align 4
%4 = icmp slt i32 %3, 100
br i1 %4, label %5, label %12
```

```
%5:
%6 = load i32* %res, align 4
%7 = load i32* %i, align 4
%8 = add nsw i32 %6, %7
store i32 %8, i32* %res, align 4
br label %9
```

```
%9:
%10 = load i32* %i, align 4
%11 = add nsw i32 %10, 1
store i32 %11, i32* %i, align 4
br label %2
```

```
ret i32 0
```

Ferramentas LLVM

- Projetos como o DragonEgg, PoLLy e NVPTX possibilitam que regiões paralelizáveis sejam traduzidas para *kernels* que possam ter sua execução lançada em GPUs.
- DragonEgg: Plugin para GCC que traduz código intermediário (GIMPLE) para a representação intermediária do LLVM (LLVM-IR).
- PoLLy: Implementação do Modelo *Polyhedral* para LLVM.
- NVPTX: *Backend* do LLVM para gerar código PTX (intermediário das GPUs da NVIDIA).

- Aplica otimizações e transformações sobre o código intermediário LLVM-IR.
- Da mesma forma que outras ferramentas, o PoLLy trabalha com (*SCoPs*).
- Na representação LLVM-IR, um SCoP é um subgrafo do grafo de fluxo de controle (CFG) que é uma região que possui um único ponto de entrada e um único ponto de saída, e é semanticamente equivalente à definição clássica de SCoP, tendo somente *FOR loops* e *IFs*.
- O PoLLy implementa um conjunto de passos que detectam e traduzem essas partes do código para a representação do modelo *polyhedral* para que possam ser aplicadas análises e transformações, e o código LLVM-IR otimizado gerado.

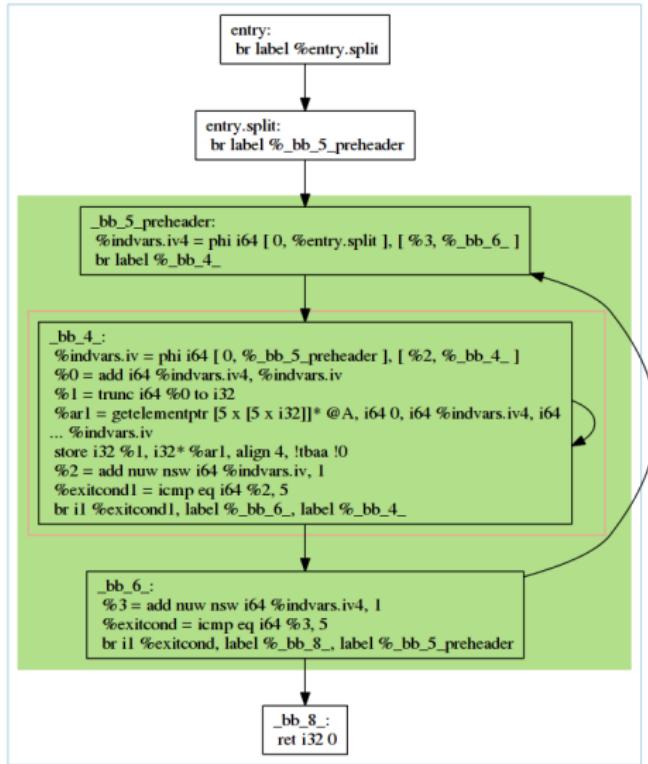
- Para aplicar transformações o PoLLy utiliza outras ferramentas como a `isl` e o `CLooG`, além de ter suporte a otimizadores externos como `PLuTo` e `PoCC`.
- Semelhante às outras ferramentas de detecção, o PoLLy exporta as definições de regiões detectadas.
- Utiliza o formato `.jscop`
- Essas definições podem ser modificadas e importadas para produzirem transformações no código.

- Exemplo de código para interchange: ($i \leftrightarrow j$):

```
1 int A[N][M];
2
3 int main() {
4     int i, j;
5
6     for (i = 0; i < N; i++) {
7         for (j = 0; j < M; j++) {
8             A[i][j] = i + j;
9         }
10    }
11    return 0;
12 }
```

- Após o código ser traduzido para LLVM-IR e preparado para o PoLLy.

PoLLy IV



- A definição do SCoP pode ser exportada (CLooG, ScopLib, jscop):

```
1 opt -load LLVMPolly.so -basicaa -polly-export-jscop  
interchange_preopt_prepare.ll
```

PoLLy VI

- Definições exportadas (arquivo .jscop):

```
1 {
2   "context" : "{   }",
3   "name" : "_bb_5_preheader => _bb_8_",
4   "statements" : [
5     {
6       "accesses" : [
7         {
8           "kind" : "write",
9           "relation" : "{ Stmt_bb_4_[i0, i1] -> MemRef_A[5
10             i0 + i1] }"
11         }
12       ],
13       "domain" : "{ Stmt_bb_4_[i0, i1] : i0 >= 0 and i0 <= 4
14             and i1 >= 0 and i1 <= 4 }",
15       "name" : "Stmt_bb_4_",
16       "schedule" : "{ Stmt_bb_4_[i0, i1] -> scattering[0, i0
17             , 0, i1, 0] }"
18     }
19   ]
20 }
```



PoLLy VII

- Alteração na ordem dos índices ($i \leftrightarrow j$):

```
1 {
2   "context" : "{   }",
3   "name" : "_bb_5_preheader => _bb_8_",
4   "statements" : [
5     {
6       "accesses" : [
7         {
8           "kind" : "write",
9           "relation" : "{ Stmt_bb_4_[i, j] -> MemRef_A[5i
10          + j] }"
11         }
12       ],
13       "domain" : "{ Stmt_bb_4_[i, j] : i >= 0 and i <= 4 and
14          j >= 0 and j <= 4 }",
15       "name" : "Stmt_bb_4_",
16       "schedule" : "{ Stmt_bb_4_[i, j] -> scattering[j, i] }"
17     }
18   ]
19 }
```



- Importação das definições modificadas:

```
1 opt -load LLVMPolly.so -basicaa -polly-import-jscop -polly-import-jscop-postfix=interchanged -polly-cloog -analyze interchange_preopt_prepare.ll
```

PoLLy IX

- É feita a leitura e o código modificado é gerado pelo CLooG:

```
1 Printing analysis 'Polly - Import Scops from JSON (Reads a .jscop file for
   each Scop)' for region: '_bb_4 => _bb_6' in function 'main':
2 Printing analysis 'Execute Cloog code generation' for region: '_bb_4 =>
   _bb_6' in function 'main':
3 Reading JSkop '_bb_5_preheader => _bb_8' in function 'main' from './main_%
   _bb_5_preheader-%_bb_8_.jskop' interchanged'.
4 Printing analysis 'Polly - Import Scops from JSON (Reads a .jscop file for
   each Scop)' for region: '_bb_5_preheader => _bb_8' in function 'main':
5   Context:
6   {
7     Statements {
8       Stmt_bb_4_
9         Domain :=
10        { Stmt_bb_4_[i0, i1] : i0 >= 0 and i0 <= 4 and i1 >= 0 and
11          i1 <= 4 };
12        Scattering :=
13          { Stmt_bb_4_[i, j] -> scattering[j, i] };
14        MustWriteAccess :=
15          { Stmt_bb_4_[i0, i1] -> MemRef_A[5i0 + i1] };
16      }
17    Printing analysis 'Execute Cloog code generation' for region: '_bb_5_
   _preheader => _bb_8' in function 'main':
18    main():
19    for (c1=0;c1<=4;c1++) {
20      for (c2=0;c2<=4;c2++) {
21        Stmt_bb_4_(c2, c1);
22      }
```

- Código gerado pelo CLooG:

```
1 main () :  
2     for (c1=0;c1<=4;c1++) {  
3         for (c2=0;c2<=4;c2++) {  
4             Stmt__bb_4_(c2,c1);  
5         }  
6     }
```

PoLLy XI

- Exemplo de *Skewing*:

```
1 {
2     "context" : "{   }",
3     "name" : "_bb_5_preheader => _bb_8_",
4     "statements" : [
5         {
6             "accesses" : [
7                 {
8                     "kind" : "write",
9                     "relation" : "{ Stmt__bb_4_[i, j] -> MemRef_A[5i
10                         + j] }"
11                 }
12             ],
13             "domain" : "{ Stmt__bb_4_[i, j] : i >= 0 and i <= 4
14                         and j >= 0 and j <= 4 }",
15             "name" : "Stmt__bb_4_",
16             "schedule" : "{ Stmt__bb_4_[i, j] -> scattering[0, i +
17                         j, 0, i, 0] }"
18         }
19     ]
20 }
```

- Importação das definições para transformação *skewing*:

```
1 opt -load LLVMPolly.so -basicaa -polly-import-jscop -polly-import-jscop-postfix=skewed -polly-cloog -analyze-interchange-preopt_prepare.ll
```

PoLLy XIII

- É feita a leitura e o código modificado é gerado pelo CLooG:

```
1 Printing analysis 'Polly - Import Scops from JSON (Reads a .jscop file for
   each Scop)' for region: '_bb_4_ => _bb_6_' in function 'main'
2 Printing analysis 'Execute Cloog code generation' for region: '_bb_4_ =>
   _bb_6_' in function 'main'
3 Reading JSkop '_bb_5_preheader => _bb_8_' in function 'main' from './main_-
   %_bb_5_preheader-%_bb_8_.jskop.skewed'
4 Printing analysis 'Polly - Import Scops from JSON (Reads a .jscop file for
   each Scop)' for region: '_bb_5_preheader => _bb_8_' in function 'main'
5 Context:
6 {
7     Statements {
8         Stmt__bb_4_
9             Domain :=
10            { Stmt__bb_4_[i0, i1] : i0 >= 0 and i0 <= 4 and i1 >= 0 and
11              i1 <= 4 };
12            Scattering :=
13                { Stmt__bb_4_[i0, i1] -> scattering[0, i0 + i1, 0, i0, 0] };
14            MustWriteAccess :=
15                { Stmt__bb_4_[i0, i1] -> MemRef_A[5i0 + i1] };
16    }
17    Printing analysis 'Execute Cloog code generation' for region: '_bb_5_
   preheader => _bb_8_' in function 'main'
18    main():
19        for (c2=0;c2<=8;c2++) {
20            for (c4=max(0,c2-4);c4<=min(4,c2);c4++) {
21                Stmt__bb_4_(c4,c2-c4);
22            }
23        }
24    }
```

- Código gerado pelo CLooG:

```
1 main () :  
2     for ( c2=0; c2<=8; c2++ ) {  
3         for ( c4=max(0 ,c2 -4) ; c4<=min ( 4 ,c2 ) ; c4++ ) {  
4             Stmt__bb_4_( c4 ,c2-c4 ) ;  
5         }  
6     }
```

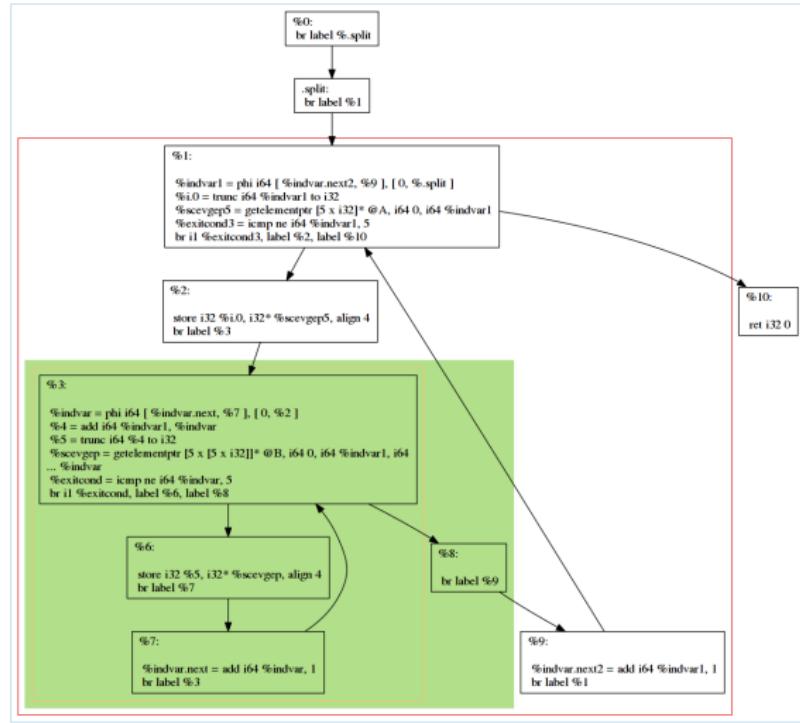
PoLLy XV

- Exemplo de código para Fissão (Fission).

```
1 #define N 5
2 #define M 5
3
4 int A[N];
5 int B[N][M];
6
7 int main() {
8     int i, j;
9     for (i = 0; i < N; i++) {
10         A[i] = i;
11         for(j = 0; j < M; j++){
12             B[i][j] = i + j;
13         }
14     }
15     return 0;
16 }
```

PoLLy XVI

- Após o código ser traduzido para LLVM-IR e preparado para o PoLLy.



- A definição do SCoP pode ser exportada (CLooG, ScopLib, jscop):

```
1 opt -load LLVMPolly.so -basicaa -polly-export-jscop  
fission_preopt_prepare.ll
```

- Definições exportadas (arquivo .jscop):

```
1 {
2     "context" : "{   }",
3     "name" : "%1 => %8",
4     "statements" : [
5         {
6             "accesses" : [
7                 {
8                     "kind" : "write",
9                     "relation" : "{ Stmt_2[i0] -> MemRef_A[i0] }"
10                }
11            ],
12            "domain" : "{ Stmt_2[i0] : i0 >= 0 and i0 <= 4 }",
13            "name" : "Stmt_2",
14            "schedule" : "{ Stmt_2[i0] -> scattering[i0, 0] }"
15        },
16        {
17            "accesses" : [
18                {
19                    "kind" : "write",
```

PoLLy XIX

```
20         "relation" : "{ Stmt_4[i0, i1] -> MemRef_B[5i0 + i1]
21             }"
22     }
23     ],
24     "domain" : "{ Stmt_4[i0, i1] : i0 >= 0 and i0 <= 4 and i1
25             >= 0 and i1 <= 4 }",
26     "name" : "Stmt_4",
27     "schedule" : "{ Stmt_4[i0, i1] -> scattering[i0, i1] }"
28 }
```

PoLLy XX

- Alteração na função de *scattering*: uma dimensão a mais.

```
1 {
2   "context" : "{ : }",
3   "name" : "%1 => %8",
4   "statements" : [
5     {
6       "accesses" : [
7         {
8           "kind" : "write",
9           "relation" : "{ Stmt_2[i0] -> MemRef_A[i0] }"
10          }
11        ],
12        "domain" : "{ Stmt_2[i0] : i0 >= 0 and i0 <= 4 }",
13        "name" : "Stmt_2",
14        "schedule" : "{ Stmt_2[i0] -> scattering[0, i0, 0] }"
15      },
16      {
17        "accesses" : [
18          {
19            "kind" : "write",
```

PoLLy XXI

```
20         "relation" : "{ Stmt_4[i0, i1] -> MemRef_B[5i0 + i1] }"
21             "
22     ],
23     "domain" : "{ Stmt_4[i0, i1] : i0 >= 0 and i0 <= 4 and i1
24         >= 0 and i1 <= 4 }",
25     "name" : "Stmt_4",
26     "schedule" : "{ Stmt_4[i0, i1] -> scattering[1, i0, i1] }"
27 }
28 }
```

- Importação das definições modificadas:

```
1 opt -load LLVMPolly.so -basicaa -polly-import-jscop -polly-
      import-jscop-postfix=fission -polly-cloog -analyze
      fission_preopt_prepare.ll
```



PoLLy XXII

- É feita a leitura e o código modificado é gerado pelo CLooG:

```
1 Printing analysis 'Polly - Import Scops from JSON (Reads a .jscop file for
   each Scop)' for region: '%3 => %6' in function 'main':
2 Printing analysis 'Execute Cloog code generation' for region: '%3 => %6' in
   function 'main':
3 Printing analysis 'Polly - Import Scops from JSON (Reads a .jscop file for
   each Scop)' for region: '%1 => %8' in function 'main':
4 Context:
5 {
6 Statements {
7 Stmt_2
8     Domain :=
9         { Stmt_2[i0] : i0 >= 0 and i0 <= 4 };
10    Scattering :=
11        { Stmt_2[i0] -> scatteriOng[0, i0, 0] };
12    MustWriteAccess :=
13        { Stmt_2[i0] -> MemRef_A[i0] };
14 Stmt_4
15     Domain :=
16         { Stmt_4[i0, i1] : i0 >= 0 and i0 <= 4 and i1 >= 0 and i1 <=
17             4 };
18     Scattering :=
19         { Stmt_4[i0, i1] -> scatteriOng[1, i0, i1] };
20     MustWriteAccess :=
21         { Stmt_4[i0, i1] -> MemRef_B[5i0 + i1] };
22 Printing analysis 'Execute Cloog code generation' for region: '%1 => %8' in
   function 'main':
```

- Código gerado pelo CLooG:

```
1 main () :  
2   for ( c2=0;c2<=4;c2++) {  
3     Stmt_2(c2);  
4   }  
5   for ( c2=0;c2<=4;c2++) {  
6     for ( c3=0;c3<=4;c3++) {  
7       Stmt_4(c2 , c3);  
8     }  
9   }
```

- Outros exemplos de aplicação de transformações podem ser encontrados na documentação do PoLLy:
[matmul: interchange + tiling].

Gerando Código para GPU

Exemplo Soma de Vetores I

- Código do exemplo soma de vetores:

```
1 int main() {
2     int i;
3     /* Inicializacao dos vetores. */
4     init_array();
5
6     /* Calculo. */
7     for (i = 0; i < N; i++) {
8         h_c[i] = h_a[i] + h_b[i];
9     }
10
11    /* Resultados. */
12    print_array();
13    check_result();
14
15    return 0;
16 }
```

Exemplo Soma de Vetores II

- Código LLVM-IR equivalente gerado pela infraestrutura do LLVM:

```
1 @h_a = common global [1024 x float] zeroinitializer, align 16
2 @h_b = common global [1024 x float] zeroinitializer, align 16
3 @h_c = common global [1024 x float] zeroinitializer, align 16
4 define void @init_array() #0 {
5 }
6 define void @print_array() #0 {
7 }
8 define void @check_result() #0 {
9 }
10 define i32 @main() #0 {
11     %1 = alloca i32, align 4
12     %i = alloca i32, align 4
13     store i32 0, i32* %1
14     call void @init_array()
15     store i32 0, i32* %i, align 4
16     br label %2
17
18 ; <label>:2           ; preds = %18, %0
19     %3 = load i32* %i, align 4
20     %4 = icmp slt i32 %3, 1024
```

Exemplo Soma de Vetores III

```
21     br i1 %4, label %5, label %21
22
23 ; <label>:5          ; preds = %2
24 %6 = load i32* %i, align 4
25 %7 = sext i32 %6 to i64
26 %8 = getelementptr inbounds [1024 x float]* @h_a, i32 0, ←
        i64 %7
27 %9 = load float* %8, align 4
28 %10 = load i32* %i, align 4
29 %11 = sext i32 %10 to i64
30 %12 = getelementptr inbounds [1024 x float]* @h_b, i32 0, ←
        i64 %11
31 %13 = load float* %12, align 4
32 %14 = fadd float %9, %13
33 %15 = load i32* %i, align 4
34 %16 = sext i32 %15 to i64
35 %17 = getelementptr inbounds [1024 x float]* @h_c, i32 0, ←
        i64 %16
36 store float %14, float* %17, align 4
37 br label %18
38
39 ; <label>:18          ; preds = %5
```

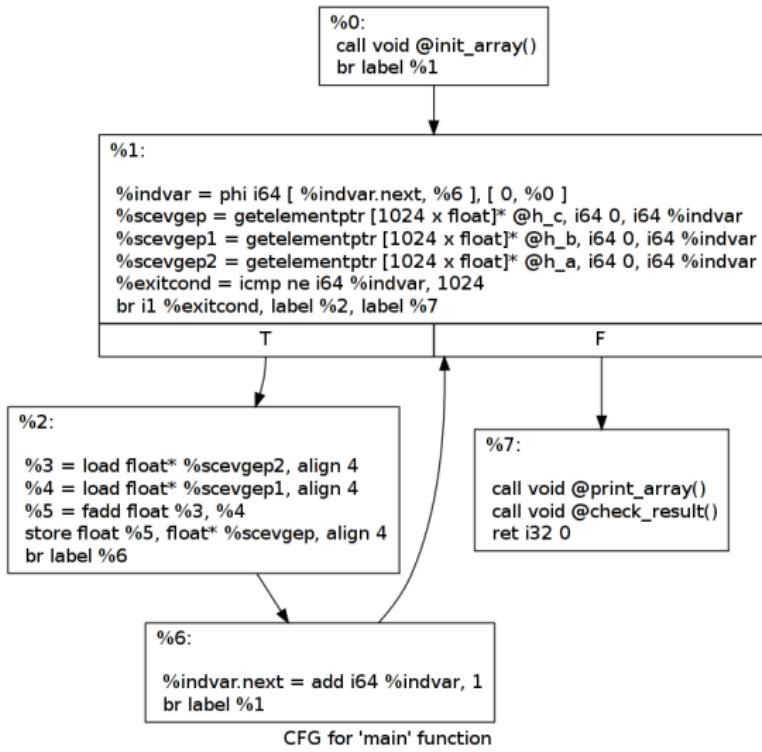
Exemplo Soma de Vetores IV

```
40 %19 = load i32* %i, align 4
41 %20 = add nsw i32 %19, 1
42 store i32 %20, i32* %i, align 4
43 br label %2
44
45 ; <label>:21           ; preds = %2
46 call void @print_array()
47 call void @check_result()
48 ret i32 0
49 }
```



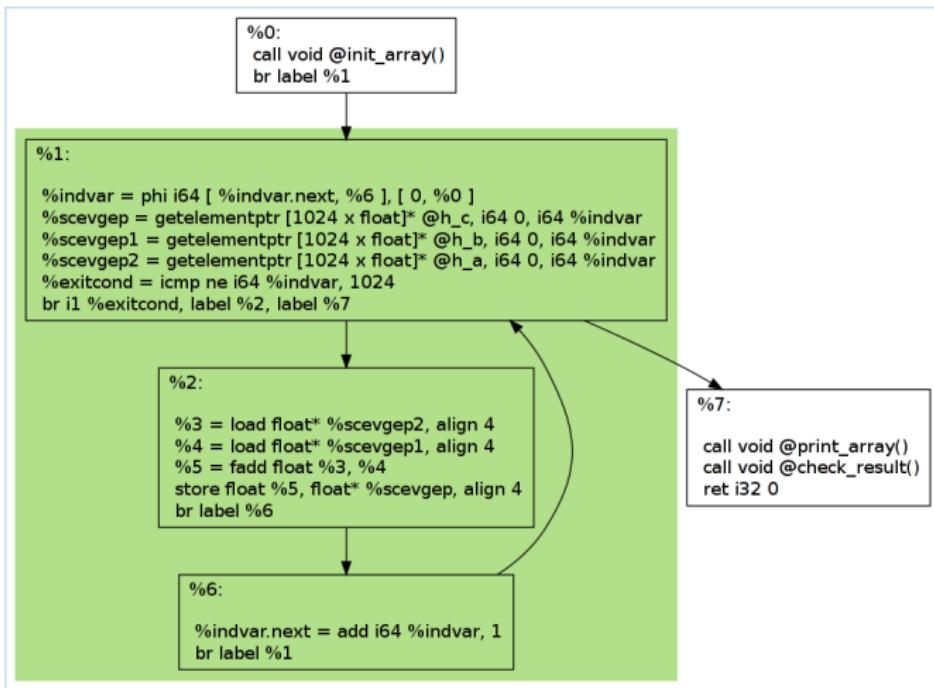
Exemplo Soma de Vetores V

O CFG do laço de repetição contido na função main:



Exemplo Soma de Vetores VI

Laço em destaque como SCoP detectado pelo PoLLy:

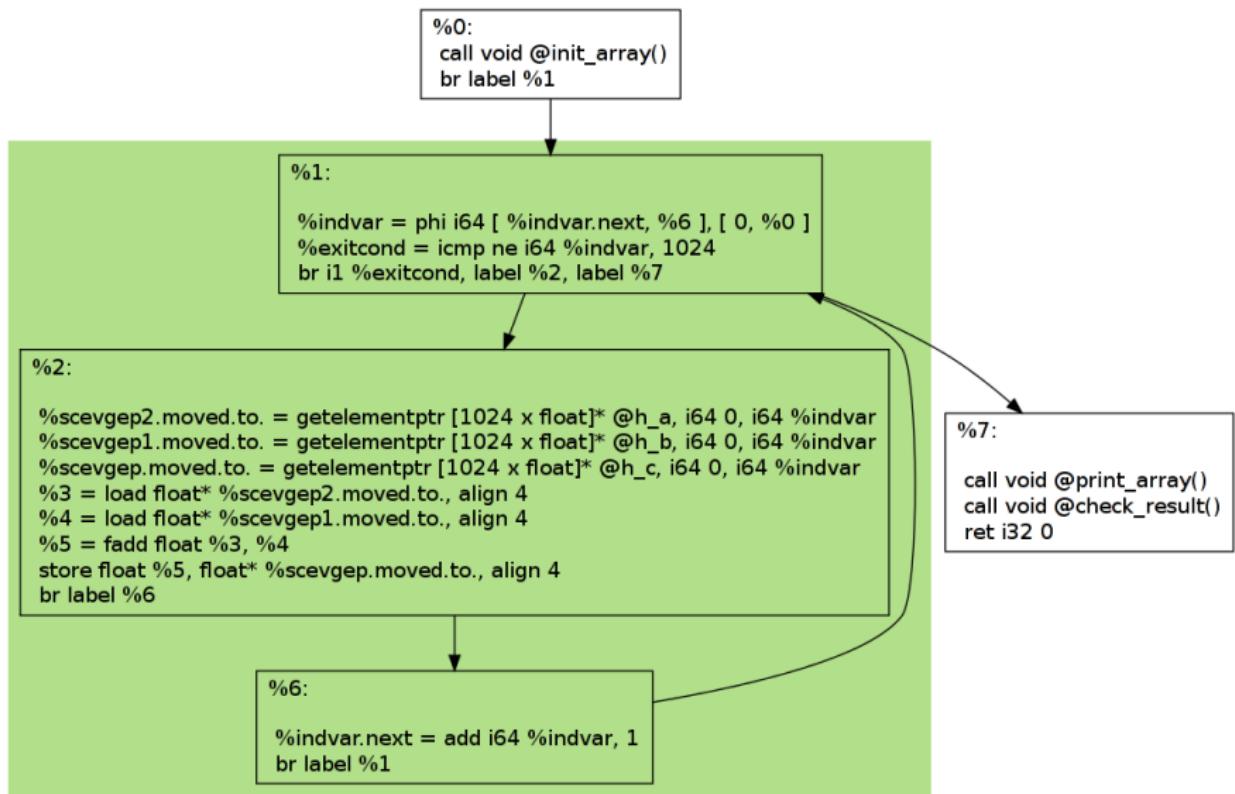


Scop Graph for 'main' function

Exemplo Soma de Vetores VII

- O PoLLy detecta as *SCoPs* do código que serão traduzidas para a representação *polyhedral*.
- O PoLLy pode ser usado para separar o código em blocos independentes.
- Visualização do código com blocos independentes que foram separados pelo PoLLy.
- O SCoP continua sendo o mesmo, porém foi alterado internamente, com a recuperação dos ponteiros para elementos dos arranjos sendo feita no início do corpo do laço.

Exemplo Soma de Vetores VIII



Scop Graph for 'main' function

Exemplo da Estrutura de um *kernel* para GPU

- Kernel para a soma de vetores:

```
1 __global__ void vecAdd(float* A, float* B, float* C, int n) {
2     int id = blockDim.x * blockIdx.x + threadIdx.x;
3     if (id < n) {
4         C[id] = A[id] + B[id];
5     }
6 }
7
8 int main() {
9     /* ... */
10    int threadsPerBlock = 256;
11    int blocksPerGrid =(N + threadsPerBlock - 1) /
12        threadsPerBlock;
13    vecAdd<<<blocksPerGrid , threadsPerBlock>>>(d_A, d_B, d_C, N);
14 }
```



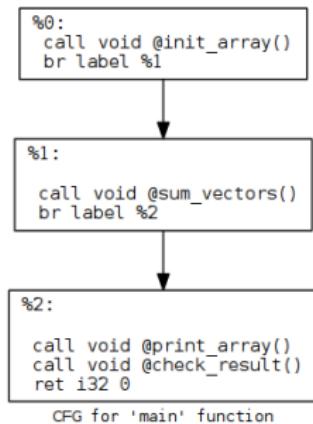
Mapeamento de identificadores no LLVM-IR

O NVPTX fornece declarações de funções que recuperam os identificadores:

CUDA	Significado	LLVM-IR para o NVPTX
threadIdx.x	Id da Thread na dim. x	declare i32 @llvm.nvvm.read.ptx.sreg.tid.x()
threadIdx.y	Id da Thread na dim. y	declare i32 @llvm.nvvm.read.ptx.sreg.tid.y()
threadIdx.z	Id da Thread na dim. z	declare i32 @llvm.nvvm.read.ptx.sreg.tid.z()
blockIdx.x	Id do Bloco na dim. x	declare i32 @llvm.nvvm.read.ptx.sreg.ctaid.x()
blockIdx.y	Id do Bloco na dim. y	declare i32 @llvm.nvvm.read.ptx.sreg.ctaid.y()
blockIdx.z	Id do Bloco na dim. z	declare i32 @llvm.nvvm.read.ptx.sreg.ctaid.z()
blockDim.x	Dimensão x do Bloco	declare i32 @llvm.nvvm.read.ptx.sreg.ntid.x()
blockDim.y	Dimensão y do Bloco	declare i32 @llvm.nvvm.read.ptx.sreg.ntid.y()
blockDim.z	Dimensão z do Bloco	declare i32 @llvm.nvvm.read.ptx.sreg.ntid.z()
gridDim.x	Dimensão x do Grid	declare i32 @llvm.nvvm.read.ptx.sreg.nctaid.x()
gridDim.y	Dimensão y do Grid	declare i32 @llvm.nvvm.read.ptx.sreg.nctaid.y()
gridDim.z	Dimensão z do Grid	declare i32 @llvm.nvvm.read.ptx.sreg.nctaid.z()
warp size	Tamanho do warp	declare i32 @llvm.nvvm.read.ptx.sreg.warpsize()

Transformação para Kernels I

- Funções de módulos LLVM-IR podem ser transformados em *kernels*.
- Para exemplificar a transformação de funções para a estrutura de *kernels*, consideremos o exemplo soma de vetores. Tanto o CFG da função principal (`main`), quanto a função `sum_vectors()`.



Transformação para Kernels II

- A correspondência entre os trechos de código LLVM-IR original e o formato para o NVPTX. Transformação da função sum_vectors para *kernel*:

```
define void @sum_vectors() #0 {  
    br label %1  
  
; <label>:1 ; preds = %6, %0  
  
%indvar = phi i64 [ %indvar.next, %6 ], [ 0, %0 ]  
%exitcond = icmp ne i64 %indvar, 1024  
br il %exitcond, label %2, label %7  
  
; <label>:2 ; preds = %1  
  
%arrayida = getelementptr [1024 x float]* @h_a, 164 0, 164 %indvar  
%arrayidb = getelementptr [1024 x float]* @h_b, 164 0, 164 %indvar  
%arrayidc = getelementptr [1024 x float]* @h_c, 164 0, 164 %indvar  
%elema = load float* %arrayida, align 4  
%elemb = load float* %arrayidb, align 4  
%sumab = fadd float %elema, %elemb  
store float %sumab, float* %arrayidc, align 4  
  
br label %6  
  
; <label>:6 ; preds = %2  
  
%indvar.next = add i64 %indvar, 1  
br label %1  
  
; <label>:7 ; preds = %1  
ret void  
}
```

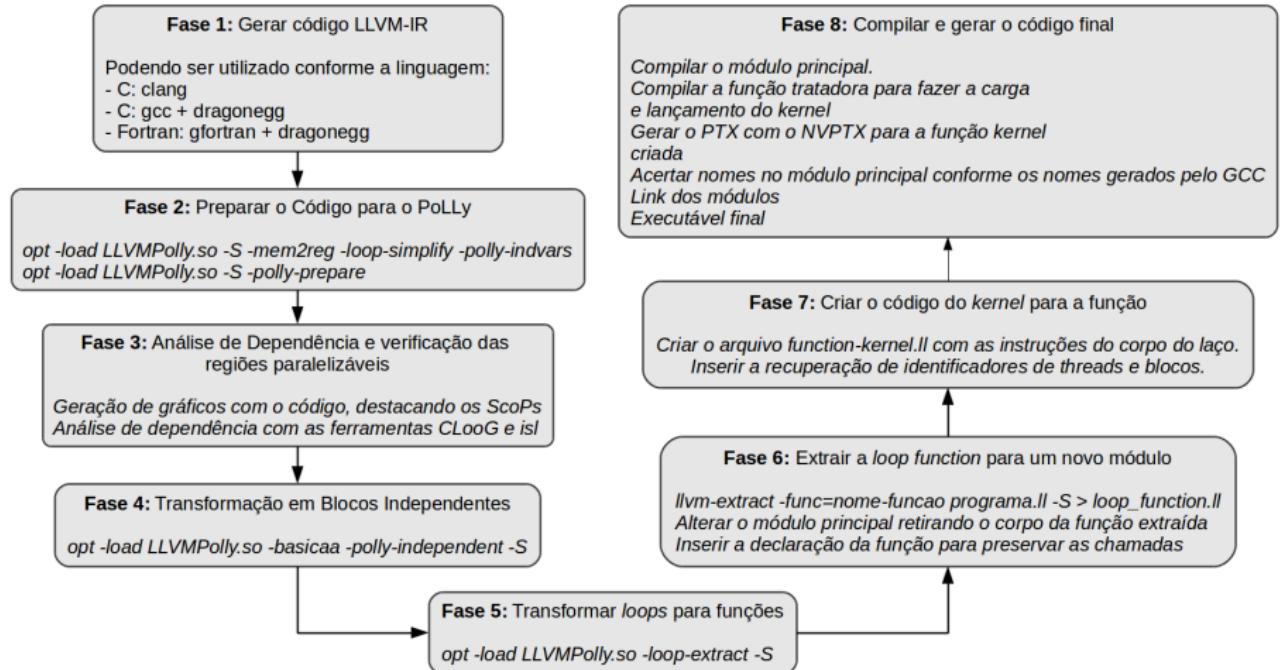
```
define ptx_kernel void @sum_kernel(float addrspace(1)* %d_a,  
                                    float addrspace(1)* %d_b,  
                                    float addrspace(1)* %d_c) uwtable {  
entry:  
  
;id = blockDim.x * blockIdx.x + threadIdx.x;  
%0 = call i32 @llvm.nvvm.read.ptx.sreg.ctaid.x()  
%1 = call i32 @llvm.nvvm.read.ptx.sreg.ntid.x()  
%2 = call i32 @llvm.nvvm.read.ptx.sreg.tid.x()  
%mul = mul i32 %0, %1  
%add = add i32 %mul, %2  
%id = sext i32 %add to i64  
  
%arrayida = getelementptr float addrspace(1)* %d_a, 164 %id  
%arrayidb = getelementptr float addrspace(1)* %d_b, 164 %id  
%arrayidc = getelementptr float addrspace(1)* %d_c , 164 %id  
%elema = load float addrspace(1)* %arrayida, align 4  
%elemb = load float addrspace(1)* %arrayidb, align 4  
%sumab = fadd float %elema, %elemb  
store float %sumab , float addrspace(1)* %arrayidc , align 4  
  
ret void  
}
```

Transformação para Kernels III

- Sequência de passos necessários para gerar, carregar o código do *kernel* e efetuar transferências:

```
1 void sum_vectors() {
2     /* Gerando o PTX. */
3     char *ptx = generatePTX(11, size);
4     /* Criando um contexto. */
5     cuCtxCreate(&cudaContext, 0, cudaDevice);
6     /* Carregando o modulo. */
7     cuModuleLoadDataEx(&cudaModule, ptx, 0, 0, 0);
8     /* Recuperando a função kernel. */
9     cuModuleGetFunction(&kernelFunction, cudaModule, "sum_kernel");
10    /* Transferindo os dados para a memória do dispositivo. */
11    cuMemcpyHtoD(devBufferA, &hostA[0], sizeof(float)*N);
12    cuMemcpyHtoD(devBufferB, &hostB[0], sizeof(float)*N);
13    /* Definição das dimensões do arranjo de threads. */
14    calculateDimensions(&blockSizeX, &blockSizeY, &blockSizeZ, &gridSizeX, &
15                        gridSizeY, &gridSizeZ);
16    /* Parâmetros da função kernel. */
17    void *kernelParams[] = { &devBufferA, &devBufferB, &devBufferC };
18    /* Lançando a execução do kernel. */
19    cuLaunchKernel(kernelFunction, gridSizeX, gridSizeY, gridSizeZ, blockSizeX
20                  , blockSizeY, blockSizeZ, 0, NULL, kernelParams, NULL);
21    /* Recuperando o resultado. */
22    cuMemcpyDtoH(&hostC[0], devBufferC, sizeof(float)*N);
23    return 0;
24 }
```

Sequência de Passos para os Próximos Exemplos I



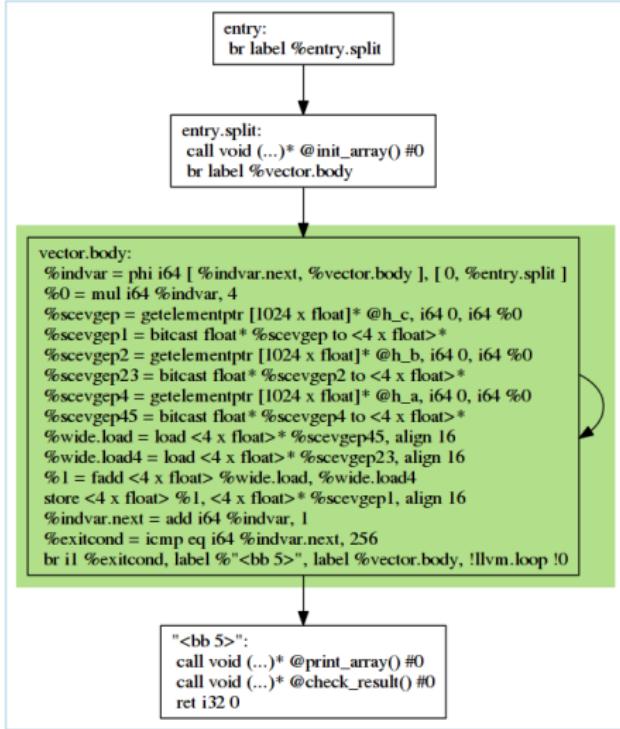
Exemplo: vectorAdd (Soma de Vetores)

```
1 float h_a[N];
2 float h_b[N];
3 float h_c[N];
4
5 int main() {
6     int i;
7     /* Inicializacao dos vetores. */
8     init_array();
9
10    /* Calculo. */
11    for (i = 0; i < N; i++) {
12        h_c[i] = h_a[i] + h_b[i];
13    }
14
15    /* Resultados. */
16    print_array();
17    check_result();
18
19    return 0;
20 }
```

Código em C.

Transformada em LLVM-IR. (Fase 1)

SCoP detectado pelo PoLLy (Fases 2, 3 e 4).



Scop Graph for 'main' function

Exemplo: vectorAdd (Soma de Vetores)



vector.body:

```
%indvar = phi i64 [ %indvar.next, %vector.body ], [ 0, %entry.split ]
%0 = mul i64 %indvar, 4
%0scevgep = getelementptr [1024 x float]* @h_c, i64 0, i64 %0
%0scevgеп1 = bitcast float* %0scevgеп to <4 x float>*
%0scevgеп2 = getelementptr [1024 x float]* @h_b, i64 0, i64 %0
%0scevgеп23 = bitcast float* %0scevgеп2 to <4 x float>*
%0scevgеп4 = getelementptr [1024 x float]* @h_a, i64 0, i64 %0
%0scevgеп45 = bitcast float* %0scevgеп4 to <4 x float>*
%wide.load = load <4 x float>* %0scevgеп45, align 16
%wide.load4 = load <4 x float>* %0scevgеп23, align 16
%1 = fadd <4 x float> %wide.load, %wide.load4
store <4 x float> %1, <4 x float>* %0scevgеп1, align 16
%indvar.next = add i64 %indvar, 1
%exitcond = icmp eq i64 %indvar.next, 256
br i1 %exitcond, label %"<bb 5>", label %vector.body, !llvm.loop !0
```



Transformação para vectoradd-kernel

```
; ModuleID = 'vectoradd.preopt.prepare.indep.blocks.loop.extract.ll'
target datalayout = "e-p:64:64-512b-11:8-8:18:8-116:16-132:32:32-164:64:64-f16:16
16- f32:32:32-32:32-64:64:64-f128:128-v64:64:64-v128:128:128-a0:0:64:64-f80:128:128-n8:16:32:64"
target triple = "x86_64-linux-gnu"

@h c = external global [1024 x float], align 32
@h b = external global [1024 x float], align 32
@h a = external global [1024 x float], align 32

; Function Attrs: nounwind
define hidden void @main_vector_body() #0 {
    newFuncRoot:
        br label @vector.body

    <bb 5>.exitStub:                                ; preds = %vector.body
        ret void

vector.body:
    %indvar_phi = phi i64 [%indvar.next, %vector.body.vector.body_crit_edge, %newFuncRoot]
    %indvar = nuw i64 %indvar_phi
    %bb_1 = blockIdx.x * blockIdx.y + threadIdx.x;
    %bb_id = tail call i32 @llvm.nvvm.read.ptx.sreg.tid.x()
    %bb_t = tail call i32 @llvm.nvvm.read.ptx.sreg.ntid.x()
    %bb_b = nuw i32 %bb_id, %bb_id
    %bb_t2 = sext i32 %bb_b, %bb_t to i64
    %indice = add i64 %bb_b, %bb_t2, 0
    br label @bb_1

    %sccevgp1 = bitcast float* %sccevgp1 to <4 x float>
    %sccevgp2 = getelementptr i1024 x float* @h.b, i64 8, 164 %bb_1
    %sccevgp3 = bitcast float* %sccevgp2 to <4 x float>
    %sccevgp4 = getelementptr i1024 x float* @h.a, i64 8, 164 %bb_1
    %sccevgp45 = bitcast float* %sccevgp4 to <4 x float>
    %wide.load = load <4 x float> %sccevgp45, align 16
    %wide.load4 = load <4 x float> %sccevgp23, align 16
    %l1 = fadd <4 x float> %wide.load, %wide.load4
    store <4 x float> %l1, %bb_1(%bb_1)
    %sccevgp1 = bitcast float* %sccevgp1 to <4 x float>
    %indvar.next = add i64 %indvar, 1
    %exitcond = icmp eq i64 %indvar.next, 256
    br i1 %exitcond, label %<bb 5>.exitStub", label %vector.body.vector.body_crit_edge, !llvm.loop !0

vector.body.vector.body_crit_edge:                  ; preds = %vector.body
    br label @vector.body
}

attributes #0 = { nounwind }

!0 = metadata !{metadata !0, metadata !1, metadata !2}
!1 = metadata !{metadata !"llvm.vectorizer.width", i32 1}
!2 = metadata !{metadata !"llvm.vectorizer.unroll", i32 1}
```

```
; ModuleID = 'vectoradd.preopt.prepare.indep.blocks.loop.extract.main_vector.body.ll'
target datalayout = "e-p:64:64-64-11:8-8:18:8-116:16-132:32:32-164:64-64-f132:32:32-64
16:64-f16:16-v32:32:32-v64:64-v128:128:128-a0:0:64:64-f80:128:128-n8:16:32:64"
target triple = "nvptx64-nvidia-cuda"

; Intrinsic to read X component of thread ID
declare i32 @llvm.nvvm.read.ptx.sreg.tid.x() readnone nounwind
declare i32 @llvm.nvvm.read.ptx.sreg.ntid.x() readnone nounwind
declare i32 @llvm.nvvm.read.ptx.sreg.ctaid.x() readnone nounwind

define ptx_kernel void @vectoradd_kernel(float* %d_a, float* %d_b, float* %d_c) uwtable {
entry:
    %id = blockIdx.x * blockIdx.y + threadIdx.x;
    %bb_id = tail call i32 @llvm.nvvm.read.ptx.sreg.tid.x()
    %bb_t = tail call i32 @llvm.nvvm.read.ptx.sreg.ntid.x()
    %bg_id = tail call i32 @llvm.nvvm.read.ptx.sreg.ctaid.x()

    %bb_b = nuw i32 %bb_id, %bb_id
    %bb_t2 = sext i32 %bb_b, %bb_t to i64
    %indice = add i64 %bb_b, %bb_t2, 0

    br label @bb_1

bb_1:                                              ; preds = %entry
    %sccevgp = getelementptr float* %d_c , 164 %indice
    %sccevgp1 = bitcast float* %sccevgp to <1 x float>
    %sccevgp2 = getelementptr float* %d_b, 164 %indice
    %sccevgp3 = bitcast float* %sccevgp2 to <1 x float>
    %sccevgp4 = getelementptr float* %d_a, 164 %indice
    %sccevgp45 = bitcast float* %sccevgp4 to <1 x float>
    %wide.load = load <1 x float> %sccevgp45, align 4
    %wide.load4 = load <1 x float> %sccevgp23, align 4
    %l1 = fadd <1 x float> %wide.load, %wide.load4
    store <1 x float> %l1, %bb_1(%bb_1)
    %sccevgp1 = bitcast float* %sccevgp1 to <1 x float>

    return:                                         ; preds = %bb_1
        ret void

    !nvvm.annotations = !(0)
    !0 = metadata !{void !(float,
                          float,
                          float)* @vectoradd_kernel, metadata !"vectoradd_kernel", i32 1}
```

Código para o vectoradd-kernel

```
; ModuleID = 'vectoradd_preopt_prepare_indep_blocks_loop_extract_main_vector.body.ll'
target datalayout = "e-p:64:64-11:8:8-18:8-116:16:16-132:32:32-164:64:f32:32:32-f64
:f64:64-v16:16-v32:32:32-v64:64-v128:128-n16:32:64"
target triple = "%nvptx64-nvidia-cuda"

; Intrinsic to read X component of thread ID
declare i32 @llvm.nvvm.read.ptx.sreg.tid.x() readnone nounwind
declare i32 @llvm.nvvm.read.ptx.sreg.ntid.x() readnone nounwind
declare i32 @llvm.nvvm.read.ptx.sreg.ctaid.x() readnone nounwind

define ptx_kernel void @vectoradd_kernel(float* %d_a, float* %d_b, float* %d_c) uwtable {
entry:
    ;%id = blockDim.x * blockIdx.x + threadIdx.x;
    %t_id = tail call i32 @llvm.nvvm.read.ptx.sreg.tid.x();
    %b_id = tail call i32 @llvm.nvvm.read.ptx.sreg.ntid.x();
    %g_id = tail call i32 @llvm.nvvm.read.ptx.sreg.ctaid.x();
    %g_b = mul i32 %b_id, %b_id
    %g_b_t = add i32 %g_b, %t_id
    %g_b_t2 = sext i32 %g_b_t to i64
    %indice = add i64 %g_b_t2, 0
    br label %bb_1

bb_1:
    ; preds = %entry
    %%scevgep = getelementptr float* %d_c, 164 %indice
    %%scevgep1 = bitcast float* %%scevgep to <1 x float>*
    %%scevgep2 = getelementptr float* %d_b, 164 %indice
    %%scevgep23 = bitcast float* %%scevgep2 to <1 x float>*
    %%scevgep4 = getelementptr float* %d_a, 164 %indice
    %%scevgep45 = bitcast float* %%scevgep4 to <1 x float>*
    %%wide.load = load <1 x float>* %%scevgep45, align 4
    %%wide.load4 = load <1 x float>* %%scevgep23, align 4
    %ab = fadd <1 x float> %ab , <1 x float>* %%scevgep1 , align 4
    store <1 x float> %ab , <1 x float>* %%scevgep1 , align 4

    br label %return

return:
    ret void
}

!nvvm.annotations = {!{!0}
!0 = metadata !{void (float*, float, float)* @vectoradd_kernel, metadata !"vectoradd_kernel", i32 1}
```

entry:

```
%t_id = tail call i32 @llvm.nvvm.read.ptx.sreg.tid.x()
%b_id = tail call i32 @llvm.nvvm.read.ptx.sreg.ntid.x()
%g_id = tail call i32 @llvm.nvvm.read.ptx.sreg.ctaid.x()
%g_b = mul i32 %g_id, %b_id
%g_b_t = add i32 %g_b, %t_id
%g_b_t2 = sext i32 %g_b_t to i64
%indice = add i64 %g_b_t2, 0
br label %bb_1
```

bb_1:

```
%%scevgep = getelementptr float* %d_c, i64 %indice
%%scevgep1 = bitcast float* %%scevgep to <1 x float>*
%%scevgep2 = getelementptr float* %d_b, i64 %indice
%%scevgep23 = bitcast float* %%scevgep2 to <1 x float>*
%%scevgep4 = getelementptr float* %d_a, i64 %indice
%%scevgep45 = bitcast float* %%scevgep4 to <1 x float>*
%%wide.load = load <1 x float>* %%scevgep45, align 4
%%wide.load4 = load <1 x float>* %%scevgep23, align 4
%ab = fadd <1 x float> %ab , <1 x float>* %%scevgep1 , align 4
store <1 x float> %ab , <1 x float>* %%scevgep1 , align 4
br label %return
```

return:
ret void

Código PTX para o vectoradd-kernel

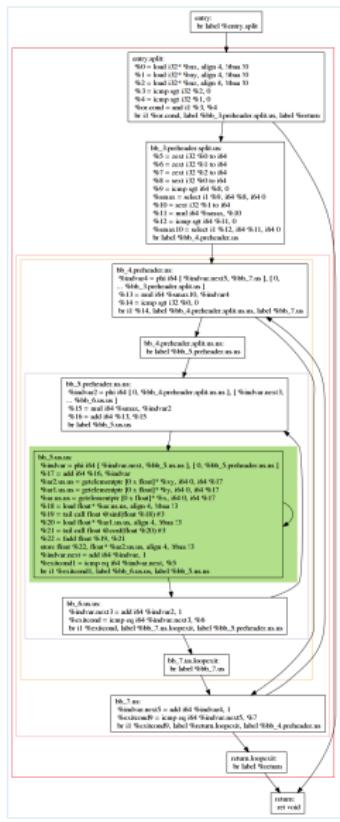
```
//  
// Generated by LLVM NVPTX Back-End  
  
.version 3.1  
.target sm_20  
.address_size 64  
  
.globl    vectoradd_kernel          // @vectoradd_kernel  
.visible .entry vectoradd_kernel(  
    .param .u64 vectoradd_kernel_param_0,  
    .param .u64 vectoradd_kernel_param_1,  
    .param .u64 vectoradd_kernel_param_2  
)  
{  
    .reg .f32      %f<4>;  
    .reg .s32      %r<5>;  
    .reg .s64      %rl<8>;  
  
// BB#0:  
    ld.param.u64   %rl1, [vectoradd_kernel_param_0];  
    mov.u32        %rl1, %tid.x;  
    ld.param.u64   %rl2, [vectoradd_kernel_param_1];  
    mov.u32        %rl2, %ntid.x;  
    ld.param.u64   %rl3, [vectoradd_kernel_param_2];  
    mov.u32        %rl3, %ctaid.x;  
    mad.lo.s32    %rl4, %r3, %r2, %rl1;  
    mul.wide.s32  %rl4, %r4, 4;  
    add.s64        %rl5, %rl3, %rl4;  
    add.s64        %rl6, %rl2, %rl4;  
    add.s64        %rl7, %rl1, %rl4;  
    ld.f32         %f1, [%rl7];  
    ld.f32         %f2, [%rl6];  
    add.f32        %f3, %f1, %f2;  
    st.f32         [%rl5], %f3;  
    ret;  
}
```



Exemplo: *sincos* (Cálculo de Seno e Cosseno)

```
1 subroutine sincos_function(nx, ny,
    nz, x, y, xy)
2
3 implicit none
4
5 integer, intent(inout) :: nx, ny,
    nz
6 real, intent(in) :: x(nx, ny, nz),
    y(nx, ny, nz)
7 real, intent(inout) :: xy(nx, ny,
    nz)
8
9 integer :: i, j, k
10
11 do k = 1, nz
12     do j = 1, ny
13         do i = 1, nx
14             xy(i, j, k) = sin(x(i, j, k))
15                 + cos(y(i, j, k))
16     enddo
17 enddo
18
19 end subroutine sincos_function
```

Rotina em Fortran + Código Principal em C.
Transformada em LLVM-IR. (Fase 1)
SCoP detectado pelo PoLLy (Fases 2, 3 e 4).



Exemplo: *sincos* (Cálculo de Seno e Cosseno)



bb_5.us.us:

```
%indvar = phi i64 [ %indvar.next, %bb_5.us.us ], [ 0, %bb_5.preheader.us.us ]
%17 = add i64 %16, %indvar
%ar2.us.us = getelementptr [0 x float]* %xy, i64 0, i64 %17
%ar1.us.us = getelementptr [0 x float]* %y, i64 0, i64 %17
%ar.us.us = getelementptr [0 x float]* %x, i64 0, i64 %17
%18 = load float* %ar.us.us, align 4, !tbaa !3
%19 = tail call float @sinf(float %18) #3
%20 = load float* %ar1.us.us, align 4, !tbaa !3
%21 = tail call float @cosf(float %20) #3
%22 = fadd float %19, %21
store float %22, float* %ar2.us.us, align 4, !tbaa !3
%indvar.next = add i64 %indvar, 1
%exitcond1 = icmp eq i64 %indvar.next, %5
br i1 %exitcond1, label %bb_6.us.us, label %bb_5.us.us
```



bb_6.us.us:

Transformação para sincos-kernel

```
define hidden void __sincos_function_bb_4_preheader_us(164 %smax10, 132, 164 %smax,
[0 x float]* %yy, [0 x float]* %xx, 164, 164, 164) #1 {
newFuncRoot;
br label %bb_4.preheader.us

return.loopexit.exitStub:           ; preds = %bb_7.us
ret void

bb_4.preheader.us:                 ; preds = %bb_7.us.bb_4.preheader.us.crit_edge, %newFuncRoot
hindvar = phi 164 [%indvar.next3, %bb_7.us.bb_4.preheader.us.crit_edge], [%newFuncRoot]
%y = mul 164 %smax10, %indvar
%5 = icmp sgt 132 %y, 0
br 11 %5, label %bb_4.preheader.split.us.us, label %bb_4.preheader.us.bb_7.us.crit_edge

bb_4.preheader.us.bb_7.us.crit_edge: ; preds = %bb_4.preheader.us
br label %bb_7.us

bb_4.preheader.split.us.us:         ; preds = %bb_4.preheader.us
br label %bb_5.preheader.us.us

bb_5.preheader.us.us:              ; preds = %bb_6.us.us.bb_5.preheader.us.us.crit_edge, %bb_4.preheader.split.us.us
hindvar2 = phi 164 [0, %bb_4.preheader.split.us.us], [%indvar.next3, %bb_6.us.us.bb_5.preheader.us.us.crit_edge]
%y2 = add 164 %y, %indvar2
%7 = add 164 %y2
br label %bb_5.us.us

bb_5.us.us:                       ; preds = %bb_5.us.us.bb_5.us.us.crit_edge, %bb_5.preheader.us.us
hindvar = phi 164 [%indvar.next, %bb_5.us.us.bb_5.us.us.crit_edge], [0, %bb_5.preheader.us.us]

hindvar2 = getelementptr [0 x float]* %yy, 164 0, 164 %8
har1.us.us = getelementptr [0 x float]* %yy, 164 0, 164 %8
har1.us.us = getelementptr [0 x float]* %yy, 164 0, 164 %8
%9 = load float* %yy, align 4, !tbaa !9
%10 = tail call float @sin(float %9) #8
%11 = load float* %yy, align 4, !tbaa !9
%12 = tail call float @cos(float %11) #8
%13 = fadd float %10, %12
store float %13, float* %yy, align 4, !tbaa !9

%indvar.next = add 164 %indvar, 1
%exitcond1 = icmp eq 164 hindvar.next, 1
br 11 %exitcond1, label %bb_6.us.us, label %bb_5.us.us.bb_5.us.us.crit_edge

bb_5.us.us.bb_5.us.us.crit.edge:   ; preds = %bb_5.us.us
br label %bb_5.us.us

bb_6.us.us:                       ; preds = %bb_5.us.us
hindvar.next3 = add 164 hindvar2, 1 ; preds = %bb_6.us.us
%exitcond = icmp eq 164 hindvar.next3, %2
br 11 %exitcond, label %bb_7.us.loopexit, label %bb_6.us.us.bb_5.preheader.us.us.crit_edge

bb_6.us.us.bb_5.preheader.us.us.crit.edge: ; preds = %bb_6.us.us
br label %bb_5.preheader.us.us

bb_7.us.loopexit:                 ; preds = %bb_6.us.us
br label %bb_7.us

bb_7.us:                          ; preds = %bb_7.us.loopexit, %bb_4.preheader.us.bb_7.us.crit.edge
hindvar.next5 = add 164 hindvar4, 1
%exitcond5 = icmp eq 164 hindvar.next5, %3
br 11 %exitcond5, label %return.loopexit.exitStub, label %bb_7.us.bb_4.preheader.us.us.crit.edge

bb_7.us.bb_4.preheader.us.us.crit.edge: ; preds = %bb_7.us
br label %bb_4.preheader.us
}
```

```
define ptx.kernel void __sincos_kernel(164 %nx, 164 %ny, 164 %nz, float* %d_xy, float* %d_xy, float* %d_xy) writable(%d_xy)

entry:
: threadIdx.x;
%t_idx = tail call 132 @l1v.mvnm.read.ptx.sreg.ctaid.x()
%t_idy = tail call 132 @l1v.mvnm.read.ptx.sreg.ctaid.y()
%t_idz = tail call 132 @l1v.mvnm.read.ptx.sreg.ctaid.z()

: blockIdx;
%b_idx = tail call 132 @l1v.mvnm.read.ptx.sreg.ntid.x()
%b_idy = tail call 132 @l1v.mvnm.read.ptx.sreg.ntid.y()
%b_idz = tail call 132 @l1v.mvnm.read.ptx.sreg.ntid.z()

: gridDim;
%g_dimx = tail call 132 @l1v.mvnm.read.ptx.sreg.nctaid.x()
%g_dimy = tail call 132 @l1v.mvnm.read.ptx.sreg.nctaid.y()
%g_dimz = tail call 132 @l1v.mvnm.read.ptx.sreg.nctaid.z()

%multi1 = mul 132 %b_idx, %b_dimx
%multi2 = mul 132 %b_idx, %b_dimy, %multi1
%multi3 = mul 132 %b_idx, %b_dimz, %multi2
%multi4 = mul 132 %t_idy, %multi3
%multi5 = mul 132 %t_idz, %multi4
%multi6 = mul 132 %t_idz, %multi5
%multi7 = mul 132 %t_idy, %multi6
%multi8 = mul 132 %t_idy, %b_dimx
%multi9 = add 132 %multi10, %t_idx
%multi10 = add 132 %multi9, %t_idy
%multi11 = add 132 %multi10, %t_idz
%multi12 = sub 132 %multi11, %t_idz
%multi13 = sub 132 %multi12, %t_idy
%multi14 = sub 132 %multi13, %t_idx
%indice = sext 132 %multi14 to 164

: indice maximo: (%multi14)+(my*%multi2) + nz.
%nx_x_nx = mul 164 %nx, %nx
%nx_x_nx_x_my = mul 164 %nx_x_nx, %ny
%ny_x_nz = mul 164 %ny, %nz
%ny_x_nx_x_my = add 164 %nx_x_nx, %nz
%ind_max = add 164 %nx_x_nx_x_my, %ny_x_nz_nz

br label %<bb_2>

*bb_2*:
%8 = icmp ult 164 %indice, %ind_max
br 11 %8, label %<bb_3>, label %<bb_6>

*bb_3*:
%9 = tail call float @l1v.mvnm.sinapprox.ptz.ffloat(%d_xy)
%10 = tail call float @l1v.mvnm.cosapprox.ptz.ffloat(%d_xy)
%cos_y_el = load float* %d_xy, align 4, !tbaa !10
%el_y = load float* %d_xy, align 4, !tbaa !10
%cos_y_el = tail call float @l1v.mvnm.cosapprox.ptz.ffloat(%d_xy)
%sin_y_el = fadd float %sin_y_el, float* %cos_y_el
store float %sin_y_el, float* %d_xy, align 4, !tbaa !9
br label %<bb_6>

*bb_6*:
br label %return

return:
ret void
}; preds = %<bb_6>
```

Código para o sincos-kernel

```

#define ptx_kernel void @sincos_kernel(%164 %nx, %164 %ny, %164 %nz, float* %d_x,
                                    float* %d_y, float* %d_xy) wtable {
entry:
    ; threadIdx.x
    %t_idx = tail call i32 @llvm.nvvm.read.ptx.sreg.tid.x()
    %t_idy = tail call i32 @llvm.nvvm.read.ptx.sreg.tid.y()
    %t_idz = tail call i32 @llvm.nvvm.read.ptx.sreg.tid.z()

    ; blockIdx.x
    %b_idx = tail call i32 @llvm.nvvm.read.ptx.sreg.ctaid.x()
    %b_idy = tail call i32 @llvm.nvvm.read.ptx.sreg.ctaid.y()
    %b_idz = tail call i32 @llvm.nvvm.read.ptx.sreg.ctaid.z()

    ; blockDim.x
    %b_dimx = tail call i32 @llvm.nvvm.read.ptx.sreg.ntid.x()
    %b_dimy = tail call i32 @llvm.nvvm.read.ptx.sreg.ntid.y()
    %b_dimz = tail call i32 @llvm.nvvm.read.ptx.sreg.ntid.z()

    ; gridDim.x
    %g_dimx = tail call i32 @llvm.nvvm.read.ptx.sreg.nctaid.x()
    %g_dimy = tail call i32 @llvm.nvvm.read.ptx.sreg.nctaid.y()
    %g_dimz = tail call i32 @llvm.nvvm.read.ptx.sreg.nctaid.z()

    %mul1 = mul i32 %b_dimy, %b_dimx
    %mul2 = mul i32 %b_dimz, %mul1
    %mul3 = mul i32 %g_dimx, %mul2
    %mul4 = mul i32 %b_idy, %mul3

    %mul7 = mul i32 %b_idx, %mul2

    %mul9 = mul i32 %t_idx, %mul1

    %mul10 = mul i32 %t_idy, %mulx

    %sconv1 = add i32 %mul10, %t_idx
    %sconv2 = add i32 %mul10, %sconv1
    %sconv3 = add i32 %mul10, %sconv2
    %sconv4 = add i32 %mul10, %sconv3
    %indice = sext i32 %sconv4 to i64

    ; indice maximo: (%nx*%ny)+ (%ny*%nz) + %z.
    %max_xy = mul i64 %nx, %ny
    %max_x_nx_x_my = mul i64 %max_xy, %nx
    %my_X_nz = mul i64 %ny, %nz
    %my_X_nz_S_nz = add i64 %my_X_nz, %nz
    %ind_max = add i64 %max_xy * %nx_x_my, %my_X_nz_S_nz

    br label "%<bb 2>"

<bb 2>:                                ; preds = %entry
    %0 = icmp ult i64 %indice, %ind_max
    br i1 %0, label "%<bb 3>", label "%<bb 6>"

<bb 3>:                                ; preds = %<bb 2>
    %y_index = sin(i32(%index));           ; preds = %<bb 2>
    %var_us.us = getelementptr float* %d_xy, i64 %indice
    %var_el = load float* %var_us.us, align 4, !tbaa !0
    %sin_x_el = tail call float @llvm.nvvm.sin.approx.ftz.f(float %x_el)
    %y_el = load float* %var_us.us, align 4, !tbaa !0
    %cos_x_el = tail call float @llvm.nvvm.cos.approx.ftz.f(float %y_el)
    %sin_cos = fadd float %sin_x_el, %cos_x_el
    %sin_cos = fadd float %sin_cos, float* %var2.us.us, align 4, !tbaa !0
    br label "%<bb 6>"

<bb 6>:                                ; preds = %<bb 2>
    br label %return

return:
    ret void
}

```

```

entry:
    %t_idx = tail call i32 @llvm.nvvm.read.ptx.sreg.tid.x()
    %t_idy = tail call i32 @llvm.nvvm.read.ptx.sreg.tid.y()
    %t_idz = tail call i32 @llvm.nvvm.read.ptx.sreg.tid.z()
    %b_idx = tail call i32 @llvm.nvvm.read.ptx.sreg.ctaid.x()
    %b_idy = tail call i32 @llvm.nvvm.read.ptx.sreg.ctaid.y()
    %b_idz = tail call i32 @llvm.nvvm.read.ptx.sreg.ctaid.z()
    %b_dimx = tail call i32 @llvm.nvvm.read.ptx.sreg.mid.x()
    %b_dimy = tail call i32 @llvm.nvvm.read.ptx.sreg.mid.y()
    %b_dimz = tail call i32 @llvm.nvvm.read.ptx.sreg.mid.z()
    %g_dimx = tail call i32 @llvm.nvvm.read.ptx.sreg.nctaid.x()
    %g_dimy = tail call i32 @llvm.nvvm.read.ptx.sreg.nctaid.y()
    %g_dimz = tail call i32 @llvm.nvvm.read.ptx.sreg.nctaid.z()
    %mul1 = mul i32 %b_dimy, %b_dimx
    %mul2 = mul i32 %b_dimz, %mul1
    %mul3 = mul i32 %g_dimx, %mul2
    %mul4 = mul i32 %b_idy, %mul3
    %mul7 = mul i32 %b_idx, %mul2
    %mul9 = mul i32 %t_idx, %mul1
    %mul10 = add i32 %mul10, %t_idx
    %sconv1 = add i32 %mul10, %sconv1
    %sconv2 = add i32 %mul10, %sconv2
    %sconv3 = add i32 %mul10, %sconv3
    %sconv4 = add i32 %mul10, %sconv4
    %indice = sext i32 %sconv4 to i64
    %max_xy = mul i64 %max_xy
    %max_x_nx_x_my = mul i64 %max_xy, %nx
    %my_X_nz = mul i64 %my_X_nz, %nz
    %ind_max = add i64 %max_xy * %nx_x_my, %my_X_nz_S_nz
    br label "%<bb 2>"

<bb 2>:
    %0 = icmp ult i64 %indice, %ind_max
    br i1 %0, label "%<bb 3>", label "%<bb 6>"

<bb 3>:
    %var2.us.us = getelementptr float* %d_xy, i64 %indice
    %var1.us.us = getelementptr float* %d_xy, i64 %indice
    %var0.us.us = getelementptr float* %d_xy, i64 %indice
    %var_el = load float* %var0.us.us, align 4, !tbaa !1
    %sin_x_el = tail call float @llvm.nvvm.sin.approx.ftz.f(float %x_el)
    %y_el = load float* %var1.us.us, align 4, !tbaa !1
    %cos_x_el = tail call float @llvm.nvvm.cos.approx.ftz.f(float %y_el)
    %sin_cos = fadd float %sin_x_el, %cos_x_el
    store float %sin_cos, float* %var2.us.us, align 4, !tbaa !1
    br label "%<bb 6>"

<bb 6>:
    br label %return

```

Proposta

- Estudos preliminares mostram que as ferramentas não exploram efetivamente todos os recursos disponíveis nas plataformas.
- Não consideram a existência de múltiplos elementos de processamento e nem a combinação deles em uma solução híbrida.
- Comportamento que evidencia o uso não eficiente de recursos, que são subutilizados ou desprezados na geração de código e em sua execução.
- Existe a necessidade de um ambiente de execução que explore cenários heterogêneos, com o uso de elementos *multicore* e multiGPU.

Proposta

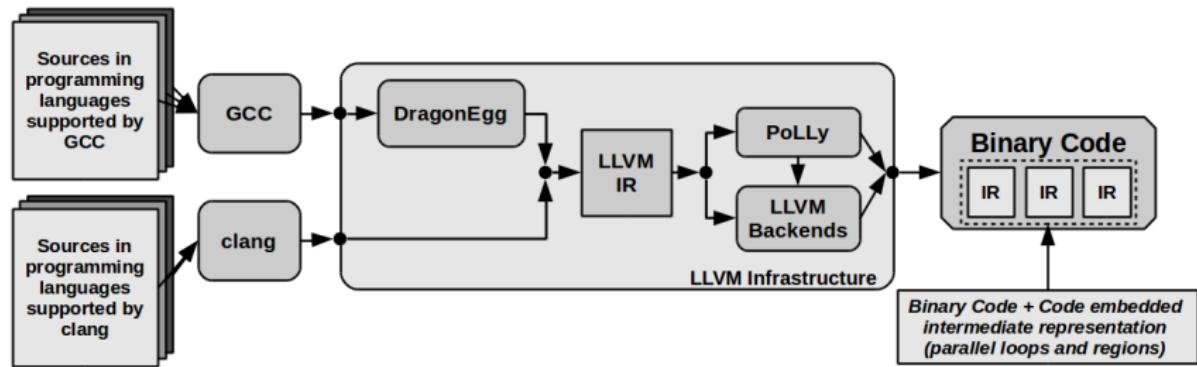
- Nossa proposta está diretamente relacionada com a abordagem de paralelização automática.
- Queremos explorar o potencial paralelismo latente das aplicações, considerando plataformas heterogêneos e como seus recursos podem ser usados eficientemente.
- Execução combinada de múltiplas aplicações, na tentativa de explorar o paralelismo entre aplicações distintas para a ocupação de recursos.

Proposta I

- Nossa proposta divide-se em duas partes fundamentais:
 - *Ferramenta de Compilação*: que prepare o código de entrada para o ambiente de execução.
 - *Ambiente de Execução*: que explore elementos heterogêneos e que seja capaz de escalonar tarefas de múltiplas aplicações para a execução em cenários híbridos, *multicore* e multiGPU. Ajustar as dimensões dos *kernels* aos recursos disponíveis.

Ferramenta de Compilação I

- Como queremos servir aplicações de código legado, que muitas vezes são escritas nas linguagens Fortran e C e como estas linguagens são suportadas pelo compilador GCC, o *plugin* DragonEgg será usado na tradução de código intermediário GCC (GIMPLE) para o LLVM-IR.
- Fluxo do processo de compilação:



Ferramenta de Compilação II

- O processo de compilação se inicia a partir do código fonte das aplicações.
- Este passo do processo de compilação permite trazer código escrito em diferentes linguagens para a representação LLVM-IR.
- A ferramenta deve ser capaz de detectar regiões paralelizáveis, como laços de repetição, e aplicar a essas regiões otimizações e transformações que favoreçam a paralelização. [Nesta fase o PoLLy será usado para detectar regiões paralelas].
- Iremos transformar essas regiões e isolar cada uma delas dentro do corpo de uma função.
- Os módulos contendo essas funções serão extraídos e mantidos em código intermediário (LLVM-IR) e serão embarcados no código da aplicação em um segmento de código do tipo .rodata.
- As regiões paralelizáveis serão mantidas em código intermediário embarcado.

Ambiente de Execução I

- A utilização de código intermediário possibilita a transformação e a geração de código final (em tempo de execução ou antecipadamente) conforme os dispositivos presentes.
- Versões do código podem ser geradas e a versão apropriada executada conforme o dispositivo alvo.
- Se o código for para ser executado em CPU, o módulo escrito em LLVM-IR, pode ser carregado e executado via JIT do LLVM.
- Se o objetivo for executar o código em GPU, o módulo LLVM-IR será transformado em *kernel* (*kernelize*) e traduzido para PTX pelo NVPTX ou utilizando a biblioteca libNVVM, permitindo o carregamento e a execução de código PTX no modelo *just in time* (JIT).

Kernel Dimensions Fitting

Ajuste automático nas dimensões de *kernels* considerando os recursos arquiteturais. Aplicar transformações a laços para transferir iterações para os *ids* do arranjo de *grids* e blocos de *threads*. Desenrolar laço para aumentar a computação por *thread*, na tentativa de melhorar a taxa de ocupação dos multiprocessadores.

Migração de Tarefas

Migração de Tarefas com *checkpoint*. Divisão de *kernels* em *subkernels* com a utilização de paralelismo dinâmico para lançar a execução na sequência adequada intercalando com *callbacks* para devolver o controle ao *host*.

Obrigado!

- rogerioag@utfpr.edu.br
- rag@ime.usp.br

