

# Refinamiento Adaptivo de Código

César Sabater



20 de febrero de 2017

Presentación de Tesina

- Muchas aplicaciones de **cómputo intensivo** realizan **cálculos aproximados**
- Algunas Razones:
  - para simular objetos o fenómenos del mundo real
  - trabajan con sensores limitados en precisión
  - están sometidas a un deadline de tiempo
  - calculan resultados preliminares

Algunos ejemplos de estas aplicaciones son:

- Simulaciones Físicas
- Sensores de entrada, algoritmos de procesamiento
- Decodificación de Video en tiempo real
- Predicción de Terremotos
- Aplicaciones de exploración de geofísica

Normalmente, el desarrollo de estos algoritmos puede dividirse en **dos etapas**:

- Desarrollo de un kernel de computo ideal
  - ajuste del algoritmo
  - debugueo
- Optimización a una versión de producción
  - ajustando el nivel de precisión para realizar cálculos precisos solo donde es necesario
  - escalar al tamaño real del problema
  - satisfacer un deadline

Optimizar un kernel ideal tiene complicaciones:

- es **complejo**
- **lleva tiempo**
- conduce a **códigos menos mantenibles**
- hay que **hacerlo nuevamente** cuando hay cambios grandes en la estrategia

Esto se podría reducir con un **enfoque automático de compilación**

Existen enfoques automáticos de optimización para computo intensivo:

- manteniendo la semántica original

- paralelización
- localidad de datos
- vectorización

bien abordados por los compiladores

- modificando la semántica original

- relajación de dependencias
- modificación o eliminación de cálculos
- usando conocimiento específico de dominio

escasamente abordados por los compiladores

Para aplicaciones que calculan resultados aproximados, un enfoque automático de optimización que modifique la semántica original es adecuado siempre que asegure una **precisión aceptable**.

**Nuestro Objetivo:** Buscar formas de optimizar códigos que computan aproximaciones

- de forma **automática**
- **modificando la semántica** original
- asegurando una **precisión aceptable**



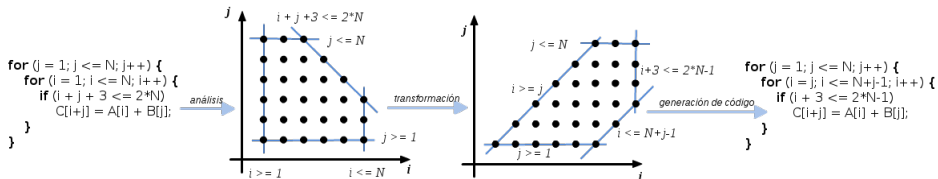
Para realizarlo utilizaremos

- El **Modelo Poliédrico** para realizar transformaciones agresivas de código
- **Conocimiento de Dominio Especifico** suministrado por el usuario
  - que guíe la transformación de código
  - mantenga una precisión aceptable
- Un enfoque **adaptivo**
  - realiza cálculos complejos solamente donde es necesario
  - ahorra computaciones que no impactan en el resultado
  - transforma el código de forma dinámica

- ➊ **Transformación de Código: Modelo Poliédrico**
- ➋ Herramienta estática: Spot
- ➌ Refinamiento Adaptivo de Código
- ➍ Experimentos con Simulación de Fluidos
- ➎ Conclusiones y Trabajo Futuro

# Modelo Poliédrico

- Es un modelo matemático-computacional para la optimización y paralelización automática de programas
- muy poderoso para la representación y transformación estructural de programas
- Representa bucles matemáticamente con **relaciones afines**
  - dominios de iteración
  - relaciones de orden
  - relaciones de acceso



# Dominios de Iteración

- **instancia de statement**: una ejecución individual de un statement  $S$
- $S(i', j')$  es la instancia de  $S$  para los valores  $i = i'$  y  $j = j'$
- el **dominio de iteración** queda definido por el conjunto de valores del vector  $\begin{pmatrix} i \\ j \end{pmatrix}$
- el dominio de iteración se puede representar con **poliedros**

## multiplicacion de polinomios

```
S1:  for (i = 0; i < 2 * N - 1; i++)  
      z[i] = 0;  
  
S2:  for (i = 0; i < N; i++)  
      for (j = 0; j < N; j++)  
          z[i+j] = x[i] + y[j];
```

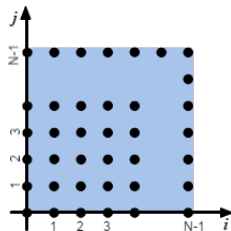


Figura: instancias de S2

# Representación poliédrica: Dominios de Iteración

inecuaciones del dominio de iteración de  $S_2$ :

- $i \geq 0$
- $i < N$
- $j \geq 0$
- $j < N$
- representación **poderosa y compacta**
- **restricciones:**
  - no hay esquema general para soportar flujo de control dinámico
  - cotas de bucles y condicionales **deben ser funciones afines** de iteradores exteriores y parámetros

ecuación matricial:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & -1 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & -1 \end{bmatrix} \begin{pmatrix} i \\ j \\ N \\ 1 \end{pmatrix} \geq \vec{0}$$

# Representación poliédrica: Relaciones de Ordenamiento

- especifican **orden temporal y espacial** de cada instancia
- **temporal**: momento de ejecución de una instancia ejecución con respecto a las otras
- **espacial**: procesador asignado
- a cada instancia se le asigna una **fecha lógica**

$$\theta_{S_1}(N, i) = \begin{pmatrix} 0 \\ i \end{pmatrix}$$

$$\theta_{S_2}(N, i, j) = \begin{pmatrix} 1 \\ i \\ j \end{pmatrix}$$

## multiplicacion de polinomios

```
S1:  for (i = 0; i < 2 * N - 1; i++)  
      z[i] = 0;  
  
      for (i = 0; i < N; i++)  
          for (j = 0; j < N; j++)  
S2:      z[i+j] = x[i] + y[j];
```

- los statements se ejecutan en **orden lexicográfico**
- si poseen la misma fecha lógica **pueden ser ejecutados en paralelo**

# Representación poliédrica: Relaciones de Ordenamiento

relaciones afines:

$$\theta_{S_1}(N) = \left\{ (i) \rightarrow \begin{pmatrix} t_1 \\ t_2 \end{pmatrix} \in \mathbb{Z} \times \mathbb{Z}^2 \left| \begin{bmatrix} -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \end{bmatrix} \begin{pmatrix} t_1 \\ t_2 \\ i \\ N \\ 1 \end{pmatrix} = \vec{0} \right. \right\},$$

$$\theta_{S_2}(N) = \left\{ \begin{pmatrix} i \\ j \end{pmatrix} \rightarrow \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} \in \mathbb{Z}^2 \times \mathbb{Z}^3 \left| \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{pmatrix} t_1 \\ t_2 \\ t_3 \\ i \\ j \\ N \\ 1 \end{pmatrix} = \vec{0} \right. \right\}$$

# Representación Poliédrica: Funciones de Acceso

- modelan accesos a memoria
- los accesos deben ser funciones afines de iteradores envolventes y parámetros fijos

$$A_{S_1,1}(N, i) = (i)$$

$$A_{S_2,2}(N, i, j) = (i + j)$$

$$A_{S_2,3}(N, i, j) = (i)$$

$$A_{S_2,4}(N, i, j) = (j)$$

```
S1:   for (i = 0; i < 2 * N - 1; i++)  
      z[i] = 0;  
  
      for (i = 0; i < N; i++)  
        for (j = 0; j < N; j++)  
S2:      z[i+j] = x[i] + y[j];
```



$$A_{S_1,1}(N) = \left\{ (i) \rightarrow (a_{S_1,1}) \in \mathbb{Z} \times \mathbb{Z} \left| \begin{bmatrix} -1 & 1 & 0 & 0 \end{bmatrix} \begin{pmatrix} a_{S_1,1} \\ i \\ N \\ 1 \end{pmatrix} = \vec{0} \right. \right\},$$

$$A_{S_2,1}(N) = \left\{ \begin{pmatrix} i \\ j \end{pmatrix} \rightarrow (a_{S_2,1}) \in \mathbb{Z}^2 \times \mathbb{Z} \left| \begin{bmatrix} -1 & 1 & 1 & 0 & 0 \end{bmatrix} \begin{pmatrix} a_{S_2,1} \\ i \\ j \\ N \\ 1 \end{pmatrix} = \vec{0} \right. \right\},$$

# Representación Poliédrica: Funciones de Acceso

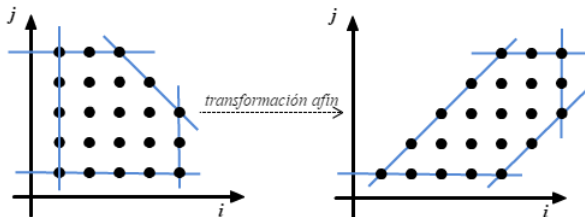
$$A_{S_2,2}(N) = \left\{ \left( \begin{pmatrix} i \\ j \end{pmatrix} \rightarrow (a_{S_2,2}) \in \mathbb{Z}^2 \times \mathbb{Z} \left[ \begin{array}{ccccc} -1 & 1 & 0 & 0 & 0 \end{array} \right] \begin{pmatrix} a_{S_2,2} \\ i \\ j \\ N \\ 1 \end{pmatrix} = \vec{0} \right) \right\},$$

$$A_{S_2,3}(N) = \left\{ \left( \begin{pmatrix} i \\ j \end{pmatrix} \rightarrow (a_{S_2,3}) \in \mathbb{Z}^2 \times \mathbb{Z} \left[ \begin{array}{ccccc} -1 & 0 & 1 & 0 & 0 \end{array} \right] \begin{pmatrix} a_{S_2,3} \\ i \\ j \\ N \\ 1 \end{pmatrix} = \vec{0} \right) \right\}.$$

# Modelo Poliédrico: Transformaciones

## transformaciones

- relaciones de ordenamiento
- dominios de iteración
- **mantienen la semántica** si respetan las dependencias de datos
  - mantener el orden de instancias **en relación de dependencia**
    - acceden a los mismos datos
    - RAW, WAR, WAW
    - RAR no es técnicamente dependencia, pero es bueno para la localidad



## generación de código

- problema: **encontrar un código eficiente que visite todos los puntos de un poliedro respetando el orden lexicográfico**
- se traduce a un problema de **escaneo de poliedros**
  - basada eliminación Fourier-Motzkin
  - programación entera perimétrica
  - método de **Quilleré, Rajopadhye y Wilde**
- extensiones de **QRW** generan códigos con **bajo costo de control**

Las herramientas de compilación poliédrica que utilizamos son:

- **OpenScop**
  - formalismo para la representación de elementos del modelo poliédrico
  - conjunto de librerías para manipular estos elementos
- **Clan**: transforma código a su representación poliédrica
- **Cloog**: generación del código con el método QRW
- **isl**: manipulación eficiente de conjuntos de enteros

- 1 Transformación de Código: Modelo Poliédrico
- 2 **Herramienta estática: Spot**
- 3 Refinamiento Adaptivo de Código
- 4 Experimentos con Simulación de Fluidos
- 5 Conclusiones y Trabajo Futuro

## Spot

- **herramienta estática** de compilación
- permite al usuario **transformar** subconjuntos del dominio de iteración
  - **reemplazando** bloques de código a ejecutar en cada instancia
  - **ignorando** la ejecución de conjuntos de instancia
- utiliza **pragmas especiales** para expresar matemáticamente sub-dominios de iteración

# Spot: pragmas

- **entrada:** bucle anidado con **pragmas** que denotan **dominios de interés**
- los **pragmas** están compuestos por
  - un *centinela* **#pragma spot**
  - una *prioridad* mayor o igual a 0
  - un *dominio de interés* en **notación isl**
  - un bloque de statement, si la *prioridad* es mayor a 0

```
#pragma spot 0 [N] ->{ [i, j] | 0<=i<=3 and 0<=j<=3}  
#pragma spot 1 [N] ->{ [i, j] | 5<=i<=8 and 2<=j<=9} A[i][j] = 1.11;  
for (i = 0; i < N; i++) {  
    for (j = 0; j < N; j++) {  
        A[i][j] = 3.14;  
    }  
}
```

- cuando la *prioridad* es 0, el sub-dominio **omitido de la ejecución**



## Salida Spot

```
if (N >= 5) {  
    for (i=0; i<=3; i++)  
        for (j=4; j<=N-1; j++)  
            A[i][j] = 3.14;  
    for (j=0; j<=N-1; j++)  
        A[4][j] = 3.14;  
}  
if (N >= 11)  
    for (i=5; i<=8; i++) {  
        for (j=0; j<=1; j++)  
            A[i][j] = 3.14;  
        for (j=2; j<=9; j++)  
            A[i][j] = 1.11;  
        for (j=10; j<=N-1; j++)  
            A[i][j] = 3.14;  
    }  
if (N <= 10)  
    for (i=5; i<=min(8, N-1); i++) {  
        for (j=0; j<=1; j++)  
            A[i][j] = 3.14;  
        for (j=2; j<=9; j++)  
            A[i][j] = 1.11;  
    }  
for (i=9; i<=N-1; i++)  
    for (j=0; j<=N-1; j++)  
        A[i][j] = 3.14;  
for (i=max(5, N); i<=8; i++)  
    for (j=2; j<=9; j++)  
        A[i][j] = 1.11;
```

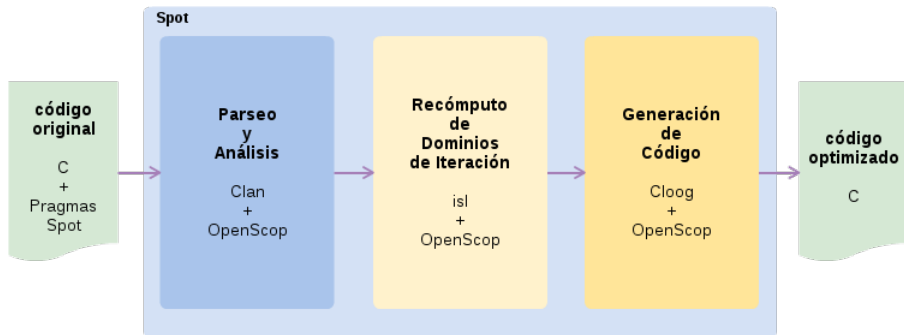
- con **bloques de calculo alternativos** en los dominios de interés con prioridad mayor a 0
- **mayor prioridad** si hay dominios que se superponen
- **omite cálculos** en regiones donde el nivel de interés es 0
- **ejecuta cálculos originales** solo donde no hay dominios de interés
- tiene **bajo costo de control**

para  $N = 10$ , con la matriz  $A$  inicializada en 0 su estado final seria:

## Salida del Codigo Generado

i / j	0	1	2	3	4	5	6	7	8	9
0	0.00	0.00	0.00	0.00	3.14	3.14	3.14	3.14	3.14	3.14
1	0.00	0.00	0.00	0.00	3.14	3.14	3.14	3.14	3.14	3.14
2	0.00	0.00	0.00	0.00	3.14	3.14	3.14	3.14	3.14	3.14
3	0.00	0.00	0.00	0.00	3.14	3.14	3.14	3.14	3.14	3.14
4	3.14	3.14	3.14	3.14	3.14	3.14	3.14	3.14	3.14	3.14
5	3.14	3.14	1.11	1.11	1.11	1.11	1.11	1.11	1.11	1.11
6	3.14	3.14	1.11	1.11	1.11	1.11	1.11	1.11	1.11	1.11
7	3.14	3.14	1.11	1.11	1.11	1.11	1.11	1.11	1.11	1.11
8	3.14	3.14	1.11	1.11	1.11	1.11	1.11	1.11	1.11	1.11
9	3.14	3.14	3.14	3.14	3.14	3.14	3.14	3.14	3.14	3.14

# Spot: Funcionamiento



- 1 Transformación de Código: Modelo Poliédrico
- 2 Herramienta estática: Spot
- 3 **Refinamiento Adaptivo de Código**
- 4 Experimentos con Simulación de Fluidos
- 5 Conclusiones y Trabajo Futuro

# Refinamiento Adaptivo de Código

Técnica para la optimización **dinámica** y **automática** de programas que **computan aproximaciones**, a través del **ahorro de calculos**.

En ingles, Adaptive Code Refinement → **ACR**

## Funcionamiento

ACR toma un código con **conocimiento de dominio** embebido

- indica **estrategias de ahorro de calculos**
- asegura **precision aceptable**

**transforma y ejecuta constantemente** versiones optimizadas del kernel

- de forma **dinamicas**
- **adaptadas** a cada estado de la ejecucion
- realiza calculos complejos **solamente donde es necesario**

# Refinamiento Adaptivo de Código

## Transformaciones

- en base al **Monitoreo** de la ejecución con una **Cuadrícula de Estado**
- **Omitien Calculos** o los reemplazan por otros **Alternativos**
- **Generan Código** eficiente con el Modelo Poliedrico (Spot)
- las realiza un **Thread Dedicado**
  - aprovecha **arquitectura multinucleo**



```
void solveOpt(int N, int D, float **X, float **W, float *C)  
{  
    int k, i, j;  
    if (N == 1) {  
        for (i=1; i<=N; i++) {  
            for (j=1; j<=N; j++) {  
                x[i][j] = (x[i][j]) + a*(x[(i)-1][j]*x[i+1][j]*x[i][j]-1)*x[i][j+1])/c;  
            }  
        }  
        if (N >= 201) {  
            for (k=1; k<=2; k++) {  
                for (i=1; i<=N; i++) {  
                    for (j=82; j<=N; j++) {  
                        x[i][j] = (x[i][j]) + a*(x[(i)-1][j]*x[i+1][j]*x[i][j]-1)*x[i][j+1])/c;  
                    }  
                }  
                for (i=61; i<=120; i++) {  
                    for (j=22; j<=N; j++) {  
                        x[i][j] = (x[i][j]) + a*(x[(i)-1][j]*x[i+1][j]*x[i][j]-1)*x[i][j+1])/c;  
                    }  
                }  
                for (i=121; i<=180; i++) {  
                    for (j=82; j<=N; j++) {  
                        x[i][j] = (x[i][j]) + a*(x[(i)-1][j]*x[i+1][j]*x[i][j]-1)*x[i][j+1])/c;  
                    }  
                }  
                for (i=141; i<=200; i++) {  
                    for (j=82; j<=N; j++) {  
                        x[i][j] = (x[i][j]) + a*(x[(i)-1][j]*x[i+1][j]*x[i][j]-1)*x[i][j+1])/c;  
                    }  
                }  
                for (i=201; i<=N; i++) {  
                    for (j=1; j<=N; j++) {  
                        x[i][j] = (x[i][j]) + a*(x[(i)-1][j]*x[i+1][j]*x[i][j]-1)*x[i][j+1])/c;  
                    }  
                }  
            }  
        }  
    }
```

Componentes:

- ➊ **Conocimiento Especifico de Dominio**
- ➋ Monitoreo del Estado de Ejecucion
- ➌ Generacion deCodigo Optimizado
- ➍ Threads de Ejecucion

## dominio de aplicacion

- código + pragmas embebidos
- **satisface restricciones** del modelo poliedrico
- **conocimiento de dominio** codificado por pragmas
  - grid
  - monitor
  - alternative
  - strategy

## Código de Simulación

```
while(true) {  
    ...  
    // lin_solve kernel  
    #pragma ACR grid(10)  
    #pragma ACR monitor(density[i][j], max, filtro)  
    #pragma ACR alternative bajo(parameter, MAX = 1)  
    #pragma ACR alternative medio(parameter, MAX = 3)  
    #pragma ACR alternative alto(parameter, MAX = 4)  
    #pragma ACR strategy direct(1, bajo)  
    #pragma ACR strategy direct(2, medio)  
    #pragma ACR strategy direct(3, alto)  
    for (k = 0; k < MAX; k++) {  
        for (i = 1; i <= N; i++) {  
            for (j = 1; j <= N; j++) {  
                lin_solve_computation(k, i, j);  
            }  
        }  
    }  
    ...  
}
```



# ACR: Conocimiento de Dominio

- **resolucion** de la cuadrícula de monitoreo:

```
#pragma ACR grid(tamaño)
```

- **porcion del estado** que vamos a monitorear:

```
#pragma ACR monitor(datos, síntesis [, filtro])
```

- **calculo alternativo** para reemplazar código original:

```
#pragma ACR alternative nombre(tipo, efecto)
```

- **estrategias** de transformación de código:

```
#pragma ACR strategy tipo(p1, p2, ...)
```

- *dinamica*: #pragma ACR strategy **direct**(*valor*, *alternativa*)
- *estatica*: #pragma ACR strategy **zone**(*área*, *alternativa*)

## Kernel de Simulacion

```
// lin_solve kernel
#pragma ACR grid(10)
#pragma ACR monitor(density[i][j], max, filtro)
#pragma ACR alternative bajo(parameter, MAX = 1)
#pragma ACR alternative medio(parameter, MAX = 3)
#pragma ACR alternative alto(parameter, MAX = 4)
#pragma ACR strategy direct(1, bajo)
#pragma ACR strategy direct(2, medio)
#pragma ACR strategy direct(3, alto)
for (k = 0; k < MAX; k++) {
    for (i = 1; i <= N; i++) {
        for (j = 1; j <= N; j++) {
            lin_solve_computation(k, i, j);
        }
    }
}
```

Componentes:

- 1 Conocimiento Especifico de Dominio
- 2 **Monitoreo del Estado de Ejecucion**
- 3 Generacion deCodigo Optimizado
- 4 Threads de Ejecucion

# ACR: Monitoreo del Estado de Ejecucion

ACR necesita obtener **informacion relevante** sobre la ejecucion

- para identificar regiones de diferente complejidad de calculo
- generar codigo adecuado para cada region
- de forma **eficiente**, para no agregar costo computacional
- resumida de forma regular, para permitir una representacion poliedrica

Para ello, utilizamos una **cuadrícula regular**

- embebida en el espacio de estado
- cada celda representa una porcion (hiper-)cubica del estado

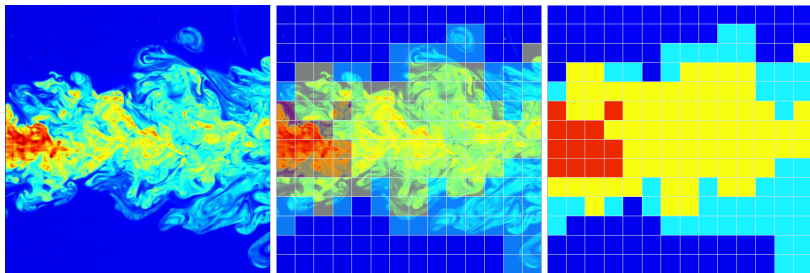
**Resolucion y Datos de Monitoreo:**

#pragma ACR **grid**(*tamaño*)

#pragma ACR **monitor**(*datos*, *síntesis* [, *filtro*])

## Obtencion de Informacion

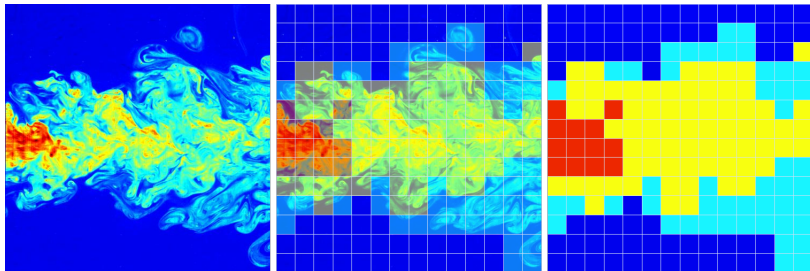
- 1 **filtro** de a valores (opcional)
- 2 **sisntesis** de informacion para cada celda
- 3 **agrupamiento** de celdas en **regiones de complejidad**



# ACR: Monitoreo del Estado de Ejecucion

## pragmas

```
#pragma grid(14)
#pragma monitor (presion[i][j],max,color)
int color(float val) {
    if      (val >= P_HIGH) return C_RED;
    else if (val >= P_MED) return C_YELLOW;
    else if (val >= P_LOW) return C_CYAN;
    else      return C_BLUE;
}
```



Componentes:

- 1 Conocimiento Especifico de Dominio
- 2 Monitoreo del Estado de Ejecucion
- 3 **Generacion deCodigo Optimizado**
- 4 Threads de Ejecucion

al principio de la ejecucion

**analysis** del kernel original con **Clan**

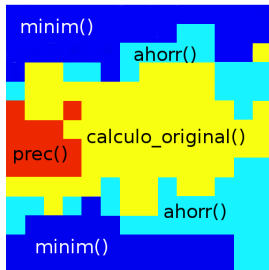
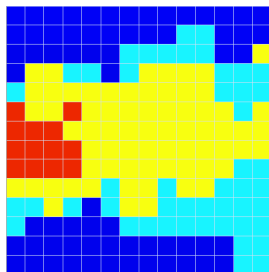
## Transformaciones Regulares

luego de un cambio significativo en el estado:

- 1 aplicacion **estrategias de transformacion**
  - omitir calculos innecesarios
  - usar bloques de codigo alternativost
- 2 **construir representacion poliedrica** de las regiones
- 3 **generacion de codigo** con Spot
- 4 **compilacion y reemplazo** del kernel que esta en ejecucion



# ACR: signacion deCodigo

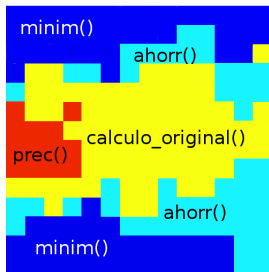
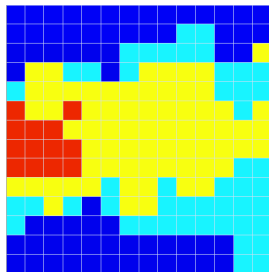


```
//kernel  
for (int i = 0; i < N; i++)  
    for (int j = 0; j < N; j++)  
        calculo_original(i,j);
```

```
// estrategias  
#pragma strategy direct(C_RED, a1)  
#pragma strategy direct(C_CYAN, a2)  
#pragma strategy direct(C_BLUE, a3)  
#pragma strategy zone  
("i < N/14 and |j - N/2| < N/14", a1)
```

```
// codigo alternativo  
#pragma alternative a1(code, prec(i,j));  
#pragma alternative a2(code, ahorr(i,j));  
#pragma alternative a3(code, minim(i,j));
```

# ACR: Asignacion deCodigo



## propiedades del codigo generado

- bajo costo de control
- **se corresponde biunivocamente** con su cuadrícula
- realiza calculos complejos **solo donde es necesario**
- **mantiene las dependencias** lo mas posible

Componentes:

- 1 Conocimiento Especifico de Dominio
- 2 Monitoreo del Estado de Ejecucion
- 3 Generacion deCodigo Optimizado
- 4 **Threads de Ejecucion**

# ACR: Threads de Ejecucion

**alivianar sobrecarga de generacion:** 2 threads

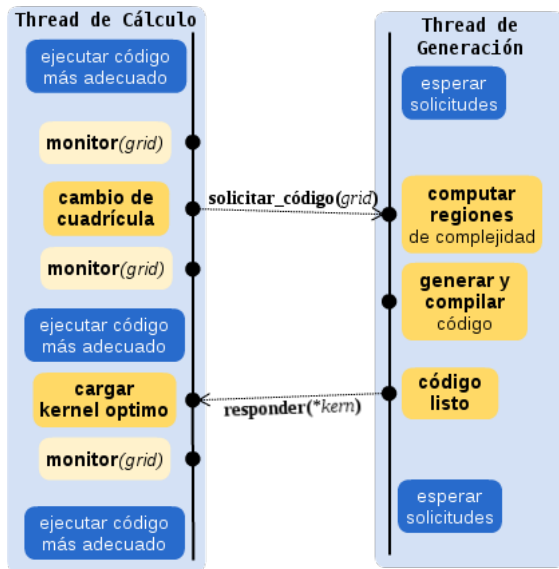
## thread de calculo

- realiza **calculos intensivos**
- **monitorea** el estado de ejecucion
- **solicita** codigo optimizado
- **recibe y carga dinamicamente** codigo optimizado
- si las optimizaciones no son adecuadas, utiliza codigo original

## thread de generacion

- 1 **recibe solicitudes** de generacion
- 2 **genera y compila** codigo optimizado
- 3 **responde las solicitudes**

# ACR: Threads de Ejecucion



- ① Transformación de Código: Modelo Poliédrico
- ② Herramienta estática: Spot
- ③ Refinamiento Adaptivo de Código
- ④ **Experimentos con Simulación de Fluidos**
- ⑤ Conclusiones y Trabajo Futuro

# Caso de Estudio: Simulacion de Fluidos

- hola

