

Aplanado eficiente de grandes modelos Modelica

Mariano Botta

FCEIA, UNR

Agosto 2015

Contenido de la charla

- 1 Motivaciones y Objetivos
- 2 Conceptos Previos
- 3 Algoritmo de aplanado
- 4 Implementación y Comparaciones
- 5 Conclusiones y Trabajo a Futuro

Contenido de la charla

- 1 Motivaciones y Objetivos
- 2 Conceptos Previos
- 3 Algoritmo de aplanado
- 4 Implementación y Comparaciones
- 5 Conclusiones y Trabajo a Futuro

Motivaciones

- Modelado, Simulación y Control en Tiempo Real con Aplicaciones en Electrónica de Potencia.
- Trabajar con sistemas a grandes escalas.
- Simulación en paralelo utilizando los métodos de cuantificación de estado.
- Aprovechar las ventajas de Modelica para describir modelos grandes.
- Las herramientas actuales no están orientadas a trabajar con modelos escalares.

Objetivos

- Mantener las características vectoriales de un modelo en las sucesivas etapas de compilación.
- Particularmente nos enfocamos en la etapa de Aplanado.
- Mantener definiciones de arreglos y ecuaciones *for* en:
 - 1 Reducción de clases.
 - 2 Resolución de conexiones.

Contenido de la charla

- 1 Motivaciones y Objetivos
- 2 Conceptos Previos**
- 3 Algoritmo de aplanado
- 4 Implementación y Comparaciones
- 5 Conclusiones y Trabajo a Futuro

Modelica

- Lenguaje de modelado orientado a objetos.
- Modelado de sistemas complejos, con componentes mecánicos, eléctricos, electrónicos, hidráulicos, térmicos, etc.
- Desarrollado por la asociación sin fines de lucro “Modelica Association”.
- Los modelos son descritos en texto plano.
- Entornos de desarrollo: OpenModelica, MathModelica, Dymola, etc.
- Librería con componentes ya definidos.

Clases

- Define un objeto.
- Son instanciadas mediante la definición de variables.
- Tienen tres secciones:
 - 1 Definiciones.
 - 2 Ecuaciones.
 - 3 Sentencias.
- Clases especializadas: *model*, *record*, *block*, *connector*, *function*, *package*

```
class X
    // Definiciones de
        variables y clases
equation
    // Ecuaciones
statements
    // Sentencias
end X;
```


Herencias de Clases

- Agrega significado semántico al modelo.
- Facilita la reutilización de código.
- Se utiliza la palabra reservada: *extends*.

La clase hijo obtiene las características del padre.

```
model OnePort
  Pin p;
  Pin n;
  Real v;
  Real i;
equation
  v = p.v - n.v;
  i = p.i;
  i = -n.i;
end OnePort;
model Capacitor
  extends OnePort;
  parameter Real C = 1;
equation
  C * der(v) = i;
end Capacitor;
```

Tipos de Variables

- **Tipos básicos:** *Real, Integer, Boolean.*
- Las clases definen un nuevo tipo.
- **Sinónimos de tipos.**
- **Prefijos de Tipo:** *flow, constant, parameter, discrete, input y output.*

```
type Current = flow Real;  
type Voltage = Real;  
connector Pin  
    Voltage v;  
    Current i;  
end Pin;
```

```
type TenPin = Pin[10];
```

```
TenPin pines;  
Pin pines2 [10];
```

Modificaciones

Aparecen en

- 1 Declaraciones de variables.
- 2 Sinónimo de tipo.
- 3 Definiciones de herencia.

Podemos

- 1 Cambiar el valor inicial de una variable.
- 2 Redefinir una variable.
- 3 Cambiar la definición de un tipo.
- 4 Anidar modificaciones.

```
package Circuits
  model CircuitX
    Capacitor cap;
    Resistor res;

    ...

  equation

  ...

end CircuitX;
model MainCircuit
  Capacitor x(C = 2);
  CircuitX co1 (cap(C = 10));
  CircuitX co2 (cap(C = 15));
end MainCircuit;
end Circuits;
```

Ecuaciones

Las ecuaciones no representan una asignación, sino igualdades.
Pueden tener expresiones complejas de ambos lados de la igualdad y expresan una relación entre las variables.

Ecuaciones de Igualdad

$$p.v - n.v = v;$$

$$p.v - n.v - v = 0;$$

$$p.v - v = n.v ;$$

Ecuaciones

Las ecuaciones no representan una asignación, sino igualdades.
Pueden tener expresiones complejas de ambos lados de la igualdad y expresan una relación entre las variables.

Ecuaciones de Igualdad

$$p.v - n.v = v;$$

$$p.v - n.v - v = 0;$$

$$p.v - v = n.v ;$$

Ecuación *for*

```
for i in 1:N loop  
    v[i] = p[i].v - n[i].v;  
end for;
```

Ecuaciones

Las ecuaciones no representan una asignación, sino igualdades.
Pueden tener expresiones complejas de ambos lados de la igualdad y expresan una relación entre las variables.

Ecuaciones de Igualdad

$$p.v - n.v = v;$$

$$p.v - n.v - v = 0;$$

$$p.v - v = n.v ;$$

Ecuación *for*

```
for i in 1:N loop  
    v[i] = p[i].v - n[i].v;  
end for;
```

N = 4

$$v[1] = p[1].v - n[1].v;$$

$$v[2] = p[2].v - n[2].v;$$

$$v[3] = p[3].v - n[3].v;$$

$$v[4] = p[4].v - n[4].v;$$

Ecuaciones *connect*

Conectores

- Son clases con ciertas restricciones.
- Se definen con el prefijo *connector*.
- No tienen ecuaciones.
- Tienen dos categorías de variables:
 - Variables de potencial. Ejemplo: presión, voltaje, etc.
 - Variables de flujo: definidas con el prefijo flow. Ejemplo: corriente, caudal, etc.

Ejemplo

Clase Pin:

- Voltaje: Variables de potencial.
- Corriente: Variables de flujo.

Ecuaciones *connect*

Ecuaciones *connect*

- Conectan dos clases del mismo tipo.
- Genera relaciones entre las variables internas de los conectores:
 - Las variables de potencial dentro de una misma conexión deben ser iguales entre sí.
 - Las variables de flujo siguen las reglas de Kirchhoff: la suma de los flujos es igual a cero.

Ecuaciones *connect*

Ecuaciones *connect*

- Conectan dos clases del mismo tipo.
- Genera relaciones entre las variables internas de los conectores:
 - Las variables de potencial dentro de una misma conexión deben ser iguales entre sí.
 - Las variables de flujo siguen las reglas de Kirchhoff: la suma de los flujos es igual a cero.

```
model LC_circuit
  Capacitor cap(v(start = 1));
  inductor ind(L = 2);
  ground gr;
equation
  connect(ind.p, cap.p);
  connect(ind.n, cap.n);
  connect(cap.n, gr.p);
end LC_circuit;
```

```
// Variables de Potencial
```

```
ind.n.v = cap.n.v;
```

```
cap.n.v = gr.p.v;
```

```
// Variables de flujo
```

```
ind.n.i + cap.n.i + gr.p.i = 0;
```

Modelo aplanado

Modelo aplanado

- 1 Modelo monolítico.
- 2 No tiene clases.
- 3 Variables de tipo básicos.
- 4 No posee ecuaciones *connect*.

Tamaños de un modelo

Tamaño de la descripción

Consideramos la cantidad de clases, variables, ecuaciones, etc. definidas.

Tamaño del sistema

Consideramos la dimensionalidad del modelo. Cantidad total de variables.

Tamaños de un modelo

Tamaño de la descripción

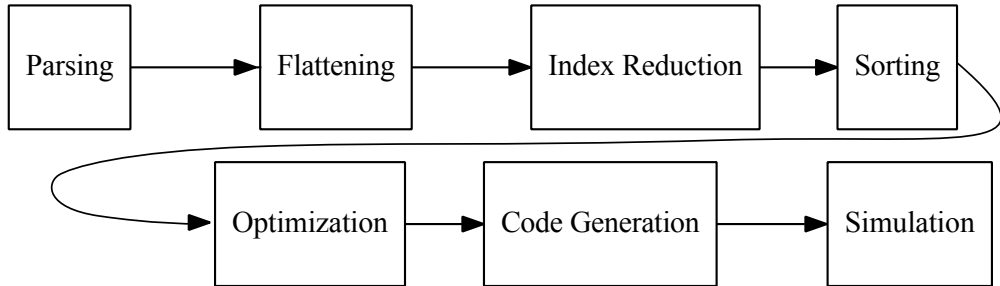
Consideramos la cantidad de clases, variables, ecuaciones, etc. definidas.

Tamaño del sistema

Consideramos la dimensionalidad del modelo. Cantidad total de variables.

```
model Big
  parameter Integer N = 1000000;
  Real a[N];
equation
  for i in 1:N loop
    a[i] = i * 10;
  end for;
end Big;
```

Simulación de Modelos Modelica



Simulación de Modelos Grandes

Problemas

Expandir las variables y ecuaciones vectorizadas implica un alto costo computacional en las siguientes etapas de compilación.

Imposibilidad de trabajar con sistemas de grandes dimensiones.

Simulación de Modelos Grandes

Problemas

Expandir las variables y ecuaciones vectorizadas implica un alto costo computacional en las siguientes etapas de compilación.

Imposibilidad de trabajar con sistemas de grandes dimensiones.

Objetivos

Algoritmo de aplanado con costo computacional constante con respecto a la dimensionalidad (cantidad de variables) del modelo.

Mantener las definiciones de arreglos y ecuaciones *for* durante toda la etapa de compilación.

Contenido de la charla

- 1 Motivaciones y Objetivos
- 2 Conceptos Previos
- 3 Algoritmo de aplanado**
- 4 Implementación y Comparaciones
- 5 Conclusiones y Trabajo a Futuro

Algoritmo de aplanado

Está dividido en dos etapas:

Reducción de composiciones

- 1 Expansión de herencia.
- 2 Simplificación de tipos.
- 3 Aplicación de modificaciones.
- 4 Reducción de instancias.

Resolución de conexiones

Descomposición de ecuaciones *connect* en ecuaciones de igualdad.

Algoritmo de aplanado

Reducción de composiciones

Algoritmo de Aplanado - Reducción de composiciones

```
Flat(C) :  
  Expand(C) ;  
  foreach v in Variables(C) :  
    t = ResolveType(v) ;  
    if isBasic(t) then  
      ChangeType(v, t) ;  
    else if isClass(t) then  
      ApplyModification(C, t, Modification(v)) ;  
      Flat(t) ;  
      RemoveComposition(C, t) ;  
      if isConnector(t) then  
        ChangeType(v, t) ;  
      else  
        Remove(t) ;  
      end if ;  
    end if ;  
  end foreach ;  
  foreach e in Equations(C) :  
    ChangeVarName(e) ;
```

Expansión de herencias - Expand

La clase hijo hereda las variables y ecuaciones del padre.

Expansión de herencias - Expand

La clase hijo hereda las variables y ecuaciones del padre.

```
model OnePort
  Pin p;
  Pin n;
  Real v;
  Real i;
equation
  v = p.v - n.v;
  i = p.i;
  i = -n.i;
end OnePort;
model Capacitor
  extends OnePort;
  parameter Real C = 1;
equation
  C * der(v) = i;
end Capacitor;
```

```
model Capacitor
  Pin p;
  Pin n;
  Real v;
  Real i;
  parameter Real C = 1;
equation
  v = p.v - n.v;
  i = p.i;
  i = -n.i;
  C * der(v) = i;
end Capacitor;
```

Simplificación de tipos - ResolveType

Determina el tipo real de una variable y sus características:

- Prefijos de Tipos.
- Definición de Arreglos.
- Presencia de modificaciones.

Simplificación de tipos - ResolveType

Determina el tipo real de una variable y sus características:

- Prefijos de Tipos.
- Definición de Arreglos.
- Presencia de modificaciones.

```
package Circuits
  model Capacitor
    extends OnePort;
    parameter Real C = 1;
    equation
      C * der(v) = i;
    end Capacitor;

    model LC_circuit
      Capacitor cap(v(start = 1));
      inductor ind(L = 2);
      Pin p1,p2,p3;
      equation
        ...
      end LC_circuit;
    end Circuits
```

Aplicación de modificaciones - ApplyModification

- 1 Expandir la clase.
- 2 Agregar las modificaciones a la variable correspondiente.

Aplicación de modificaciones - ApplyModification

- 1 Expandir la clase.
- 2 Agregar las modificaciones a la variable correspondiente.

Capacitor c (C=5,v(start=2))

```
model Capacitor
  Pin p;
  Pin n;
  Real v;
  Real i;
  parameter Real C = 1;
equation
  v = p.v - n.v;
  i = p.i;
  i = -n.i;
  C * der(v) = i;
end Capacitor;
```

Aplicación de modificaciones - ApplyModification

- 1 Expandir la clase.
- 2 Agregar las modificaciones a la variable correspondiente.

Capacitor c (C=5,v(start=2))

```
model Capacitor
  Pin p;
  Pin n;
  Real v;
  Real i;
  parameter Real C = 1;
equation
  v = p.v - n.v;
  i = p.i;
  i = -n.i;
  C * der(v) = i;
end Capacitor;
```

```
model Capacitor
  Pin p;
  Pin n;
  Real v (start=2);
  Real i;
  parameter Real C = 5;
equation
  v = p.v - n.v;
  i = p.i;
  i = -n.i;
  C * der(v) = i;
end Capacitor;
```

Reducción de instancias - RemoveComposition

RemoveComposition

- ➊ Reemplaza las instancias de clases (previamente aplanadas).
- ➋ Añade las variables y ecuaciones internas.
- ➌ Renombrar las variables añadidas para evitar colisión de nombres.
- ➍ Si la instancia está vectorizada:
 - ➊ Agrega las variables vectorizadas.
 - ➋ Encapsulamos las ecuaciones dentro de una ecuación *for*.

RemoveComposition

Variables

- 1 Agrega un prefijo al nombre de las variables: "nombreInstancia_".
- 2 Mantiene los prefijos de tipos y las definiciones de arreglos de las variables.
- 3 Agrega una nueva dimensión si la instancia estaba vectorizada.

RemoveComposition

Variables

- 1 Agrega un prefijo al nombre de las variables: "nombreInstancia_".
- 2 Mantiene los prefijos de tipos y las definiciones de arreglos de las variables.
- 3 Agrega una nueva dimensión si la instancia estaba vectorizada.

Ecuaciones

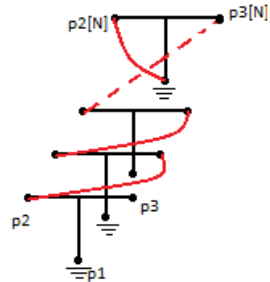
- 1 Renombra las variables que correspondan.
- 2 Reemplaza el operador "." por guiones.
- 3 Encapsula las ecuaciones en un *for* si la instancia está vectorizada.

Algoritmo de Aplanado - Reducción de composiciones

```
Flat(C) :  
  Expand(C) ;  
  foreach v in Variables(C) :  
    t = ResolveType(v) ;  
    if isBasic(t) then  
      ChangeType(v, t) ;  
    else if isClass(t) AND then  
      ApplyModification(C, t, Modification(v)) ;  
      Flat(t) ;  
      RemoveComposition(C, t) ;  
      if isConnector(t) then  
        ChangeType(v, t) ;  
      else  
        Remove(t) ;  
      end if ;  
    end if ;  
  end foreach ;  
  foreach e in Equations(C) :  
    ChangeVarName(e) ;
```

Reducción de composiciones: Ejemplo

```
package Circuits
  model LC_circuit
    Pin p1,p2,p3;
  equation
    p1.v = p2.v;
    p2.v = p3.v;
  end LC_circuit;
  model LC_line
    constant Integer N = 10;
    LC_circuit lc[N];
    ground gr[N];
  equation
    connect(lc[N].p1,lc[N].p2)
    for i in 1:N-1 loop
      connect(lc[i+1].p3,lc[i].p2);
    end for;
    for i in 1:N loop
      connect(gr.p[i],lc[i].p1);
    end for;
  end LC_line;
end Circuits;
```



Reducción de composiciones: Ejemplo

```
model LC_circuit
  Pin p1,p2,p3;
equation
  p1.v = p2.v;
  p2.v = p3.v;
end LC_circuit;

model LC_line
  constant Integer N = 10;
  LC_circuit lc[N];
  ground gr[N];
equation
  connect(lc[N].p1,lc[N].p2)
  for i in 1:N-1 loop
    connect(lc[i+1].p3,lc[i].p2);
  end for;
  for i in 1:N loop
    connect(gr.p[i],lc[i].p1);
  end for;
end LC_line;
```


Reducción de composiciones: Ejemplo

```
model LC_circuit
  Pin p1,p2,p3;
equation
  p1.v = p2.v;
  p2.v = p3.v;
end LC_circuit;

model LC_line
  constant Integer N = 10;
  LC_circuit lc[N];
  ground gr[N];
equation
  connect(lc[N].p1,lc[N].p2)
  for i in 1:N-1 loop
    connect(lc[i+1].p3,lc[i].p2);
  end for;
  for i in 1:N loop
    connect(gr.p[i],lc[i].p1);
  end for;
end LC_line;
```

```
package Circuits
  model LC_circuit
    Pin p1,p2,p3;
    flow Real p1_i,p2_i,p3_i;
    Real p1_v,p2_v,p3_v;
  equation
    p1_v = p2_v;
    p2_v = p3_v;
  end LC_circuit;
end Circuits;
```

Reducción de composiciones: Ejemplo

```
model LC_circuit
  Pin p1,p2,p3;
  flow Real p1_i,p2_i,p3_i;
  Real p1_v,p2_v,p3_v;
equation
  p1_v = p2_v;
  p2_v = p3_v;
end LC_circuit;

model LC_line
  constant Integer N = 10;
  LC_circuit lc[N];
  ground gr[N];
equation
  connect(lc[N].p1,lc[N].p2)
  for i in 1:N - 1 loop
    connect(lc[i + 1].p3,lc[i].p2);
  end for;
  for i in 1:N loop
    connect(gr.p[i],lc[i].p1);
  end for;
end LC_line;
```

Reducción de composiciones: Ejemplo

```
model LC_circuit
  Pin p1,p2,p3;
  flow Real p1_i,p2_i,p3_i;
  Real p1_v,p2_v,p3_v;
equation
  p1_v = p2_v;
  p2_v = p3_v;
end LC_circuit;

model LC_line
  constant Integer N = 10;
  LC_circuit lc[N];
  ground gr[N];
equation
  connect(lc[N].p1,lc[N].p2)
  for i in 1:N - 1 loop
    connect(lc[i + 1].p3,lc[i].p2);
  end for;
  for i in 1:N loop
    connect(gr.p[i],lc[i].p1);
  end for;
end LC_line;
```

```
package Circuits
  model LC_line
    constant Integer N = 10;
    Pin lc_p1[N], lc_p2[N], lc_p3[N];
    flow Real lc_p1_i[N], lc_p2_i[N], lc_p3_i[N];
    Real lc_p1_v[N], lc_p2_v[N], lc_p3_v[N];
    ground gr[N];
  equation
    for i in 1:N loop
      lc_p1_v[N] = lc_p2_v[N];
      lc_p2_v[N] = lc_p3_v[N];
    end for;
    connect(lc_p1[N],lc_p2[N])
    for i in 1:N - 1 loop
      connect(lc_p3[i + 1],lc_p2[i]);
    end for;
    for i in 1:N loop
      connect(gr_p[i],lc_p1[i]);
    end for;
  end LC_line;
end Circuits;
```

Algoritmo de aplanado

Resolución de conexiones

Resolución de conexiones

En un principio

- 1 Expandir las ecuaciones *for*.
- 2 Generar de un grafo a partir de las ecuaciones *connects*.
- 3 Determinar las componentes conexas del grafo generado.
- 4 Generación de ecuaciones a partir de las componentes conexas.

Resolución de conexiones

En un principio

- 1 Expandir las ecuaciones *for*.
- 2 Generar de un grafo a partir de las ecuaciones *connects*.
- 3 Determinar las componentes conexas del grafo generado.
- 4 Generación de ecuaciones a partir de las componentes conexas.

Proponemos

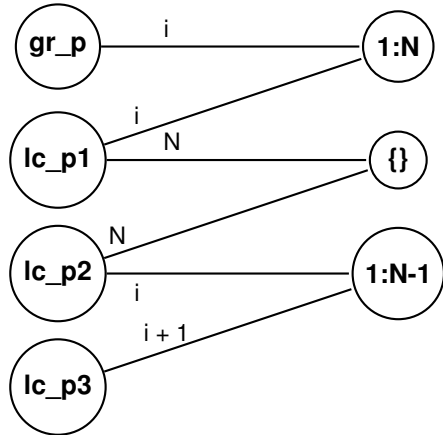
- 1 Grafo bipartito cuyos nodos y aristas están etiquetadas según las ecuaciones.
Grafo Vectorizado.
- 2 Búsqueda en profundidad modificada para determinar las componentes conexas.
- 3 Generación de ecuaciones a partir de las componentes conexas.

Grafo Vectorizado

- 1 Agregamos un nodo por cada variable de la ecuación.
- 2 Agregamos un nodo que representa a la ecuación *connect*.
- 3 Si la ecuación estaba dentro de un *for*, etiquetamos el nodo *connect* con el rango de iteración.
- 4 Agregamos dos aristas, entre cada variable y el nodo que representa al *connect*.
- 5 Por cada arista, si la variable asociada tiene un índice de acceso, agregamos esa referencia a la arista.

Grafo Bipartito: Ejemplo

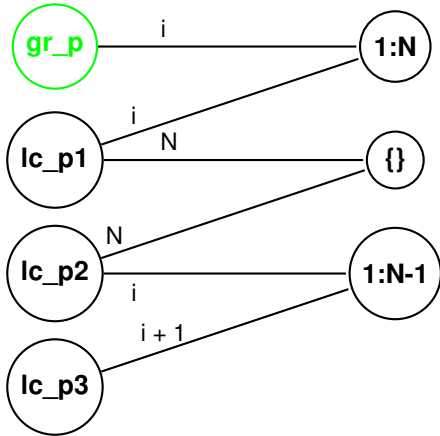
```
connect(lc_p1[N],lc_p2[N])  
  
for i in 1:N - 1 loop  
    connect(lc_p3[i + 1],lc_p2[i]);  
end for;  
  
for i in 1:N loop  
    connect(gr_p[i],lc_p1[i]);  
end for;
```



Determinación de componentes conexas

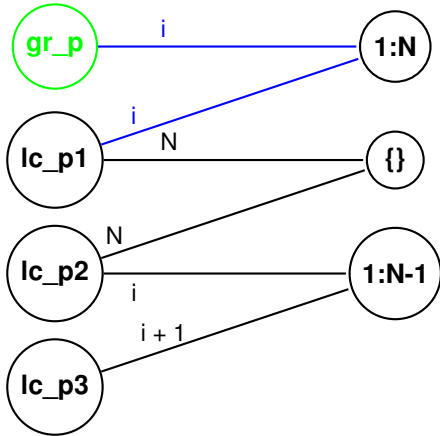
- Búsqueda en profundidad sobre el grafo.
- Cada visita a un nodo depende de la metadata del grafo.
- Múltiples visitas a un mismo nodo.
- Mantenemos referencia del intervalo de acceso al nodo.
- Visitamos un nodo si hay intersección entre intervalos.
- Podemos parte de la arista al visitar un nodo.
- Terminamos cuando no hay más intersección con otros nodos.

Determinación de componentes conexas



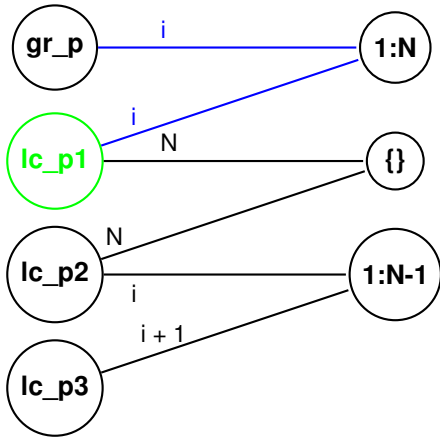
1 Arrancamos en gr_p .

Determinación de componentes conexas



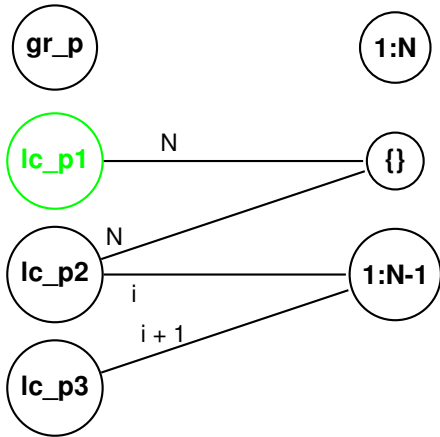
- 1 Arrancamos en gr_p .
- 2 Existe único camino y tiene intervalo $1:N$.

Determinación de componentes conexas



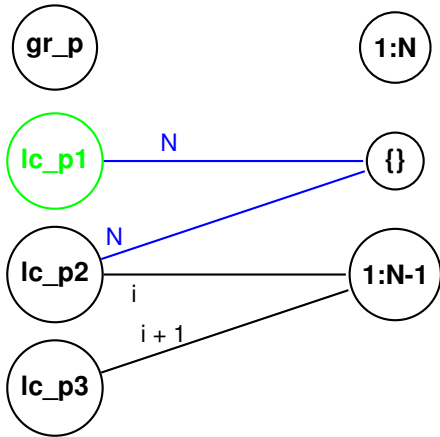
- 1 Arrancamos en gr_p .
- 2 Existe único camino y tiene intervalo $1:N$.
- 3 Visitamos lc_p1 con intervalo $1:N$.

Determinación de componentes conexas



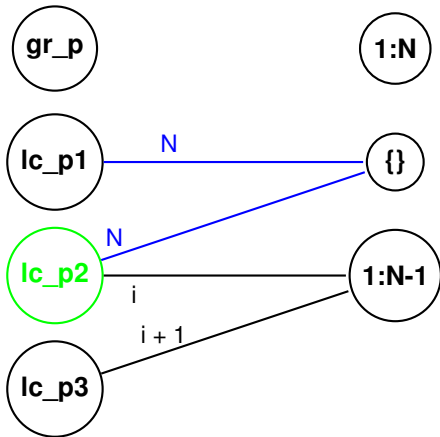
- 1 Arrancamos en gr_p .
- 2 Existe único camino y tiene intervalo $1:N$.
- 3 Visitamos lc_{p1} con intervalo $1:N$.
- 4 Podamos el grafo.

Determinación de componentes conexas



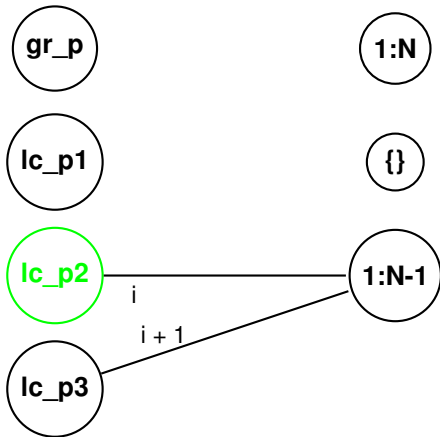
- 1 Arrancamos en *gr_p*.
- 2 Existe único camino y tiene intervalo 1:N.
- 3 Visitamos *lc_p1* con intervalo 1:N.
- 4 Podamos el grafo.
- 5 Existe un camino con índice N.

Determinación de componentes conexas



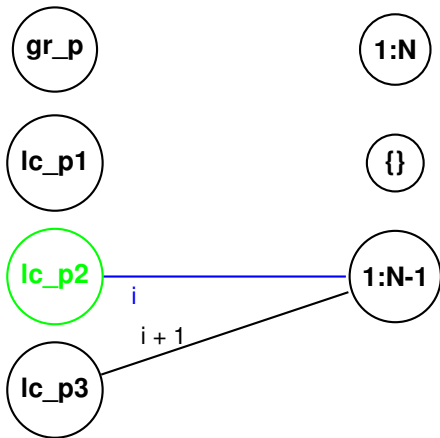
- 1 Arrancamos en gr_p .
- 2 Existe único camino y tiene intervalo $1:N$.
- 3 Visitamos lc_p1 con intervalo $1:N$.
- 4 Podamos el grafo.
- 5 Existe un camino con índice N .
- 6 Visitamos lc_p2 con índice N .

Determinación de componentes conexas



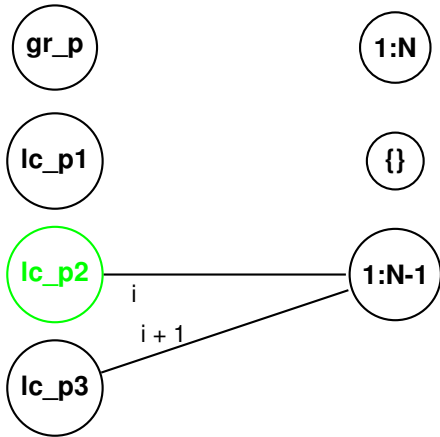
- 1 Arrancamos en gr_p .
- 2 Existe único camino y tiene intervalo $1:N$.
- 3 Visitamos lc_p1 con intervalo $1:N$.
- 4 Podamos el grafo.
- 5 Existe un camino con índice N .
- 6 Visitamos lc_p2 con índice N .
- 7 Podamos el grafo.

Determinación de componentes conexas



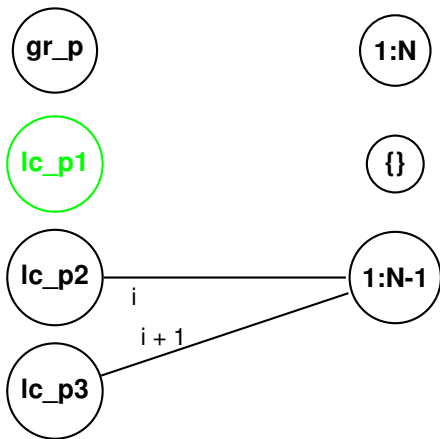
- 1 Arrancamos en gr_p .
- 2 Existe único camino y tiene intervalo $1:N$.
- 3 Visitamos lc_p1 con intervalo $1:N$.
- 4 Podamos el grafo.
- 5 Existe un camino con índice N .
- 6 Visitamos lc_p2 con índice N .
- 7 Podamos el grafo.
- 8 $N \cap 1 : N - 1 = \{\}$.
No hay más nodos para buscar.

Determinación de componentes conexas



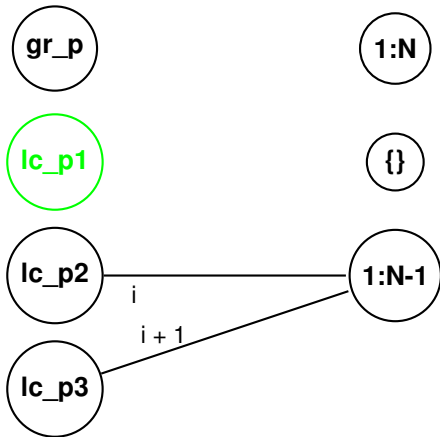
- 1 En lc_p2 la solución es:
 - $\langle lc_p2[N] \rangle$.

Determinación de componentes conexas



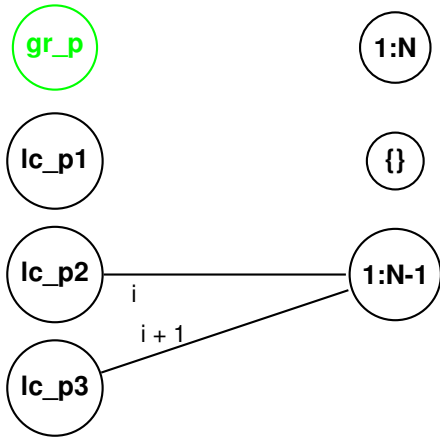
- 1 En lc_p2 la solución es:
 - $\langle lc_p2[N] \rangle$.
- 2 En lc_p1 hubo una partición de intervalos:
 - $1 : N - 1$
 - N

Determinación de componentes conexas



- 1 En lc_p2 la solución es:
 - $\langle lc_p2[N] \rangle$.
- 2 En lc_p1 hubo una partición de intervalos:
 - $1 : N - 1 \rightarrow \langle lc_p1[i] \rangle$ en $1 : N - 1$
 - $N \rightarrow \langle lc_p1[N], lc_p2[N] \rangle$

Determinación de componentes conexas



- 1 En lc_p2 la solución es:
 - $\langle lc_p2[N] \rangle$.
- 2 En lc_p1 hubo una partición de intervalos:
 - $1 : N - 1 \rightarrow \langle lc_p1[i] \rangle$ en $1 : N - 1$
 - $N \rightarrow \langle lc_p1[N], lc_p2[N] \rangle$
- 3 En gr_p las soluciones finales son:
 - $\langle gr_p[i], lc_p1[i] \rangle$ en $1 : N - 1$
 - $\langle gr_p[N], lc_p1[N], lc_p2[N] \rangle$

Generación de ecuaciones

- Cada componente conexa determina un conjunto de ecuaciones.
- Determinar el tipo de conector usado.
- Las variables de potencial deben quedar igualadas entre sí.
- Las variables de flujo deben sumar cero.

Generación de ecuaciones

- $\langle gr_p[i], lc_p1[i] \rangle$ en $1 : N - 1$.

```
for i in 1:N - 1 loop
    gr_p_v[i] = lc_p1_v[i];
    gr_p_i[i] + lc_p1_i[i] = 0;
end for;
```

Generación de ecuaciones

- $\langle gr_p[i], lc_p1[i] \rangle$ en $1 : N - 1$.

```
for i in 1:N - 1 loop
    gr_p_v[i] = lc_p1_v[i];
    gr_p_i[i] + lc_p1_i[i] = 0;
end for;
```

- $\langle gr_p[N], lc_p1[N], lc_p2[N] \rangle$.

```
gr_p_v[N] = lc_p1_v[N];
gr_p_v[N] = lc_p2_v[N];
gr_p_i[N] + lc_p2_i[N] + lc_p2_i[N] = 0;
```


Contenido de la charla

- 1 Motivaciones y Objetivos
- 2 Conceptos Previos
- 3 Algoritmo de aplanado
- 4 Implementación y Comparaciones**
- 5 Conclusiones y Trabajo a Futuro

Implementación y Comparaciones

Algoritmo de Aplanado

- 1 Implementados en C++.
- 2 Uso de las estructuras de datos de la librería Boost (AST, Árboles, Parser).
- 3 Pertenece al proyecto ModelicaCC ^a, que contiene diversas herramientas para compilar los modelos y simularlos.
- 4 Realizamos pruebas de performance en varios ejemplos variando el parámetro N.

^a<http://sourceforge.net/projects/modelicacc/>

Comparaciones con otros algoritmos

N	OpenModelica		ModelicaCC	
	Tiempo(seg)	Tamaño(bytes)	Tiempo(seg)	Tamaño(bytes)
10	3.792	33.212	0.048	3.708
100	5.632	302.374	0.052	3.723
500	19.440	818.439	0.044	3.725
1000	51.628	3.048.164	0.052	3.738
3000	393.452	9.272.336	0.044	3.740
5000	1107.732	15.496.336	0.052	3.740
10000	Error	Error	0.058	3.753

Cuadro : Tiempos de aplanado variando N para el modelo LC_line

Contenido de la charla

- 1 Motivaciones y Objetivos
- 2 Conceptos Previos
- 3 Algoritmo de aplanado
- 4 Implementación y Comparaciones
- 5 Conclusiones y Trabajo a Futuro

Conclusiones

- Desarrollamos un algoritmo de aplanado que preserva la vectorización del modelo.
- Desarrollamos un métodos para encontrar las componente conexas dentro de un grafo bipartito vectorizado.
- Implementamos ambos algoritmos en C++ dentro de la herramienta ModelicaCC.
- Realizamos pruebas tanto de ejemplos simples como de ejemplos vectoriales, concluyendo que las transformaciones aplicadas por la herramienta desarrollada llegaban al resultado correcto.
- Comparamos nuestra implementación del algoritmo de aplanado con la de la herramienta OpenModelica para distintos tamaños de modelos vectorizados, obteniendo un costo constante en función de la cantidad de variables del modelo.

Trabajo a Futuro

- Realizar pruebas con nuevos ejemplos para soportar aun más las características de Modelica.
- Adaptar el algoritmo de resolución de ecuaciones *connect* para resolver anidaciones de dos o más ecuaciones *for*.
- Aplicar un caché de modelos aplanados con el objetivo de reducir a uno la cantidad de veces que aplanamos una misma clase.
- Estudiar la posibilidad de paralelizar el aplanado de clases.

Fin

¡Muchas Gracias!
¿Preguntas?