Spatial Computing

as

Intensional Data Parallelism

Antoine Spicher
Olivier Michel
Jean-Louis Giavitto

http://mgs.spatial-computing.org

IBISC
Computer Science, Integrative Biology & Complex Systems
CNRS - University of Evry - Genopole
• Motivations: data-parallelism and spatial computing

• Intensionnal Spatial Operations

• Dataflow

• Examples

• Compilation

• Conclusions
Data Parallelism

• Parallelism and Spatial Computing: 
  \textit{if two computations occur simultaneously, they must take place at different location} 
  \Rightarrow \textit{taking space into account}

• Parallelism as an \textit{operational vs a semantic property}

• Three ways to express parallelism:
  • parallelism is expressed through the data: data parallelism
  • parallelism is expressed through the control: control parallelism
  • parallelism is expressed through a mix of data and control: pipe-line

• An alternative classification:

<table>
<thead>
<tr>
<th></th>
<th>0 INSTRUCTION COUNTER</th>
<th>1 INSTRUCTION COUNTER</th>
<th>(n) INSTRUCTIONS COUNTER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Declarative languages</td>
<td>Sequential languages</td>
<td>Concurrent languages</td>
</tr>
<tr>
<td>SCALAR</td>
<td>SISAL, ID, LAU, Actors</td>
<td>Fortran, Pascal, C</td>
<td>Adda, Occam</td>
</tr>
<tr>
<td>COLLECTION</td>
<td>Gamma, 81/2, MGS, PROTO</td>
<td>APL *Lisp, HPF, CMFortran</td>
<td>CMFortran + multi-threadings</td>
</tr>
</tbody>
</table>
The global (spatial, intensional) vs. the local (PE) view

(intensional point of view on spatially distributed objects and processes)

Global view

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>+!</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>+</td>
<td>+</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>+</td>
<td>+</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>+</td>
<td>+</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>+</td>
<td>+</td>
<td>40</td>
</tr>
</tbody>
</table>

Local view

(PE point of view)

<table>
<thead>
<tr>
<th>PE1</th>
<th>PE2</th>
<th>PE3</th>
<th>PE4</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
</tr>
</tbody>
</table>
From Arrays, Data Fields and GBF to Chain

**ARRAY**

*(Total Function)*

\[
[0, d_1] \times \ldots \times [0, d_n] \rightarrow \text{Val}
\]

**Data Field**

*(Partial Function)*

\[\mathbb{Z}^n\] \rightarrow \text{Val}

**GBF** *(Group based Fields)*

*(Partial Function)*

\[\text{Group}\] \rightarrow \text{Val}

**Chain:**

*(Partial Function)*

\[\text{Cellular complex}\] \rightarrow \text{Val}
Intentional operations

• Alpha extension
Intentional operations

- Alpha extension
- Beta reduction
Intentional operations

- Alpha extension
- Beta reduction
- Scan

\[ f \xrightarrow{\text{butterfly}} f \backslash \backslash \]

Log(size(collection))
Declarative definition

- A 8,5 program is a set of definitions:

\[
\begin{align*}
A &= B + C \\
C &= (\text{max} \ B) \ast (+\backslash \ B) \\
B[4] &= +\backslash \ (\!1)
\end{align*}
\]

\[
\begin{align*}
A &= [4, 14, 27, 44] \\
C &= 4 \ast [1, 3, 6, 10] = [4, 12, 24, 40] \\
B &= [1, 2, 3, 4]
\end{align*}
\]

- Definitions can be recursive

\[
X = 0 \# (1 + x:[3])
\]

where

- constant are polymorphic
- \# is the concatenation
- \([\] \) is the cut operation
Solving a recursive definition

- Infer the geometry
- Check that the solution is *a priori* maximal
- Compute the solution by (a smart) fixed point iteration

\[
\begin{align*}
\begin{array}{ccc}
\end{array}
&= \begin{array}{c}
0
\end{array} \# \left( 1 + \left( \begin{array}{ccc}
1+? & 1+? & 1+?
\end{array} :[3] \right) \right) \\
\begin{array}{ccc}
0 & 0 & 0
\end{array}
&= \begin{array}{c}
0
\end{array} \# \left( 1 + \begin{array}{ccc}
0 & 0 & 0
\end{array} :[3] \right) \\
\begin{array}{ccc}
0 & 1 & 0
\end{array}
&= \begin{array}{c}
0
\end{array} \# \left( 1 + \begin{array}{ccc}
0 & 1 & 0
\end{array} :[3] \right) \\
\begin{array}{ccc}
0 & 1 & 2 & 0
\end{array}
&= \begin{array}{c}
0
\end{array} \# \left( 1 + \begin{array}{ccc}
0 & 1 & 2 & 0
\end{array} :[3] \right) \\
\begin{array}{ccc}
0 & 1 & 2 & 3
\end{array}
&= \begin{array}{c}
0
\end{array} \# \left( 1 + \begin{array}{ccc}
0 & 1 & 2 & 3
\end{array} :[3] \right)
\end{align*}
\]
Inferring the geometry

\[ A = 1 \# A \]

\[ B = [1, B] \]

\[ C = 1 \# (2 \#^2 C[:2]) \]
Declarative control: stream

\[ A = 0, 1, 2, 3, 4, 3, 2, 1, 0, 1, 2, 3 \ldots \]

\[ B = 0, 1, 2, 3, 4, 3, 2, 1, 0, 1, 2, 3 \ldots \]

Introduction of *hiaton*: from data flow to synchronous data flow

\[ \text{clock}(B) = \text{true, false, true, false, true, false, true, false, true, false, true, false } \ldots \]

*clock* ... *tick*... *tock* ...
A logic of signal vs. a logic of state

A logic of signal

\[ A = B + C \]

A logic of state

\[ A = B + C \]

The result is the combination of the last seen values. A result is computed if there is a change in one input.
**Synchronous stream algebra**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1+2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clock 2</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>assuming A</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>assuming B</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C = A+B</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$ C $</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>...</td>
</tr>
<tr>
<td>A when B</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>6</td>
<td>7</td>
<td></td>
<td>9</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Recursive stream definitions

- \( X = \$X + 1 \)

\( \emptyset \) (the empty stream)

Hint: what is the initial value of the stream?

- \( X@0 = 1 \)
  \( X = \$X + 1 \)

Hint: at which pace the counter increase?

- \( X@0 = 0 \)
  \( X = \$X + 1 \) when Clock1

\( \begin{array}{cccc}
1 & 2 & \& & 4 & \& \\
\text{t} & \text{f} & \text{t} & \text{f} & \ldots \\
\end{array} \\
\text{clock(Clock1)} \)
The *wlumf*: a reactive animat

System *wlumf* = {

    glycemia@0 = 6;
    glycemia = if eating
        then 12
        else max(0, $glycemia - 1)
    when Clock

    eating@0 = false;
    eating = $hungry && environment.food;

    hungry@0 = false;
    hungry = (glycemia < 6);
}

System Environment = {
    food = ((t%2) == 0);
    t@0 = 0;
    t = $t+1 when Clock(-2);
}
Fabric = stream(collection) = collection(stream)

Fabric = stream of collection = collection of stream

(for static geometry)
Fabric = a “space-time” data
Heat diffusion in a thin rod

\[ U(0) = \ldots \]

\[ U = \alpha(\text{begin # inside})[n] + (1-2\alpha)\text{inside} + \alpha(\text{inside # end})[-n] \]

inside = \$U\text{ when Clock}
begin = 0
end = 0
Mapping and (cyclic) scheduling

The sequency graph to fold

Processors

Distribution

Width: task execution duration

Height: number of required processing elements

ok

Authorised

Split

Forbidden

The sequency graph to fold

Pattern (computation of a tick)

Scheduling

Pattern

border of ticks

border of ticks

Time
Example of a growing collection

\[ p = ((\#p \# 0 : \#p) \#^\sim (\#p \# \#p)) \text{ when } Clock 1 \]

\[ p@0 = 1 : [1, 1] \]
Conclusions

- a C compiler to a sequential architecture
- Parallel mapping and scheduling on:
  - CM
  - MPI (paragon, network of workstation)
- efficient compilation if static

- Spatial computing: YES **but**
  - **Simple** model of underlying space (but can be extended)
  - **Synchronous** time: atomic, event-driven, synchronization costs
  - **Crystalline** computation
  - **Intensional** approach = working with *spatial object as a whole*
  - NO support for amorphous computing:
    - Locality can be enforced through a tailored set of operations
    - no robustness
    - Dynamic space are difficult to handle