technology
from seed

Caravela: A Distributed Stream-based Computing Platform

Leonel Sousa, Schinichi Yamagiwa

Students involved: Diogo Antão, Gabriel Falcão and Tomás Brandão

Outline

technology
from seed

inesc id lisboa

1. Stream-based computing
2. GPUs
3. Flow-model
4. Caravela platform
5. Research future direction

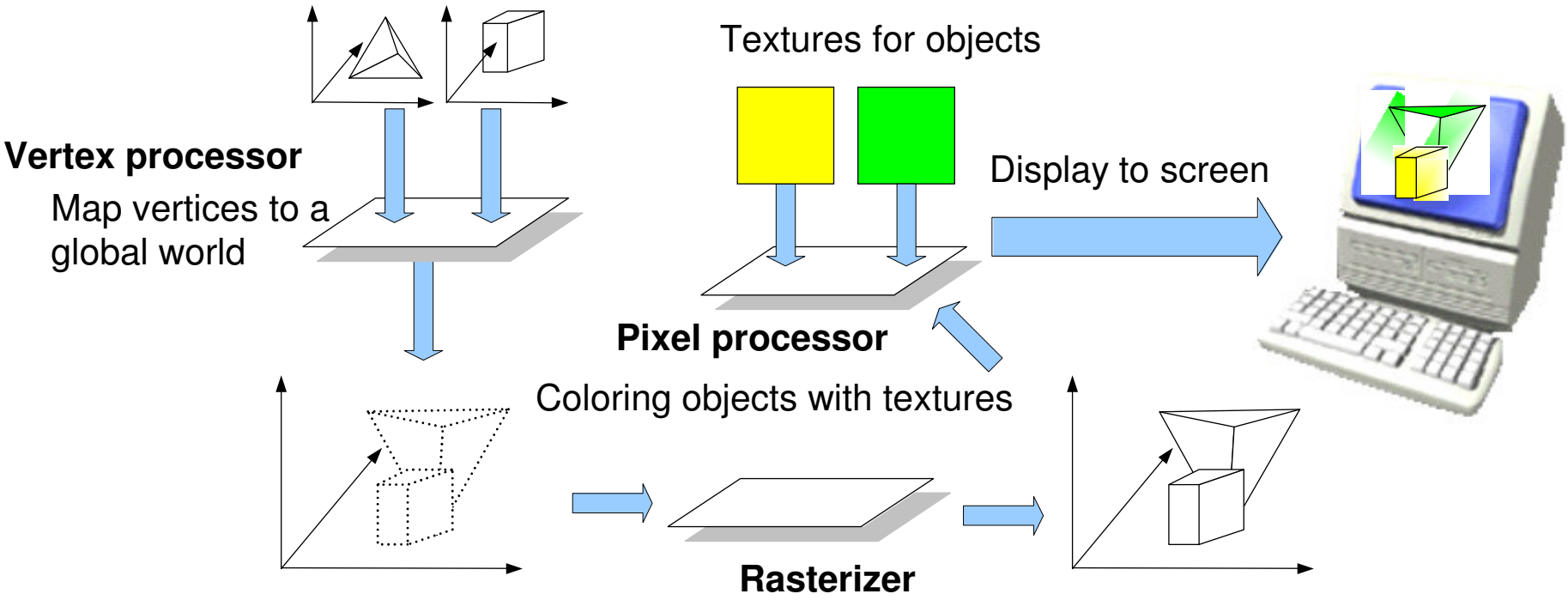
INSTITUTO SUPERIOR TÉCNICO

Stream-based computing

- Stream-based computing
  - Gathers data into a stream
  - Processes
  - Scatters the output data stream to memory
- Applications
  - Audio, Image and Video processing (AAC, JPEG2000, MPEG CoDec)
  - Numerical and Scientific Computations
  - Simulation of Fluid Flows
- Merits
  - Streams are the only resources that have to be touched by programs
  - No stall in processing
  - Hardware available to speed up processing

Graphics Processing Unit

Graphics Processing Unit



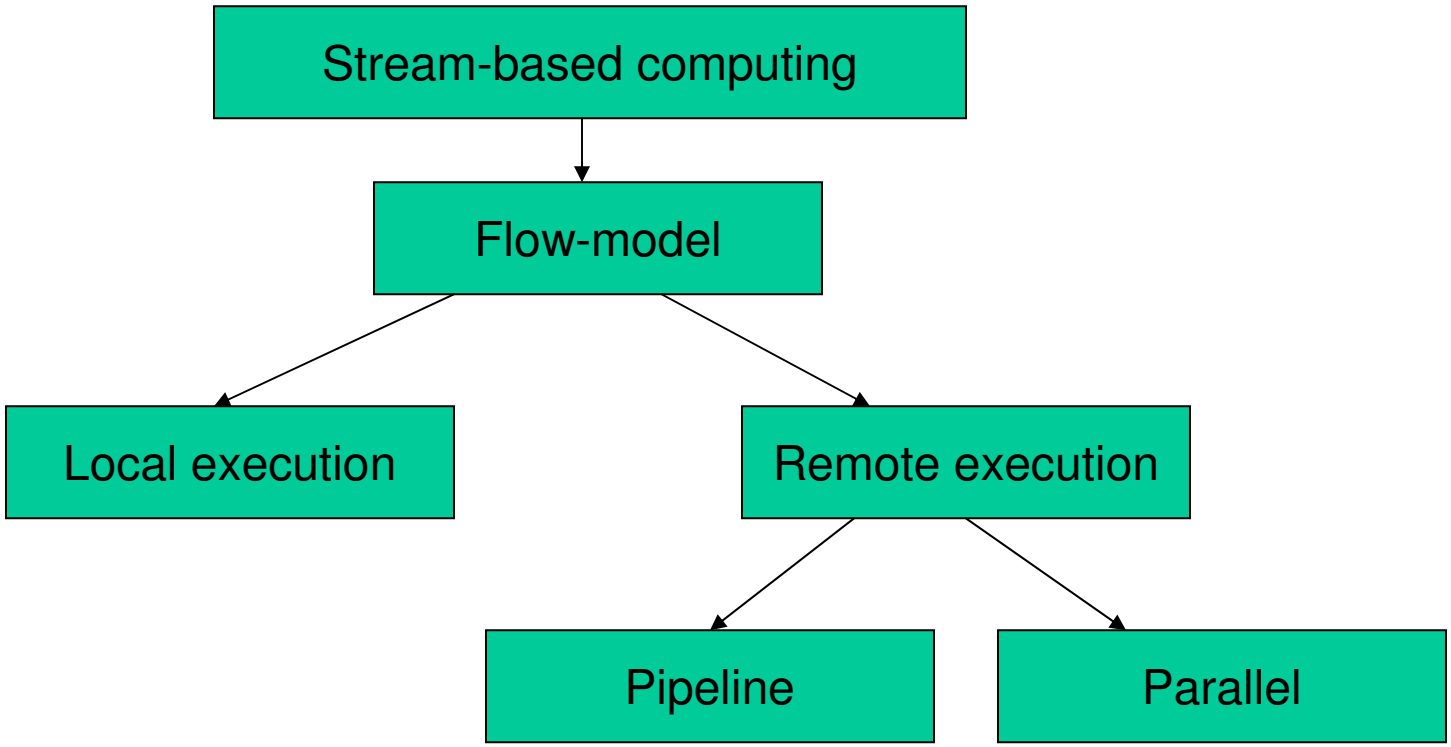
GPGPU

General Purpose processing on Graphics Processing Unit

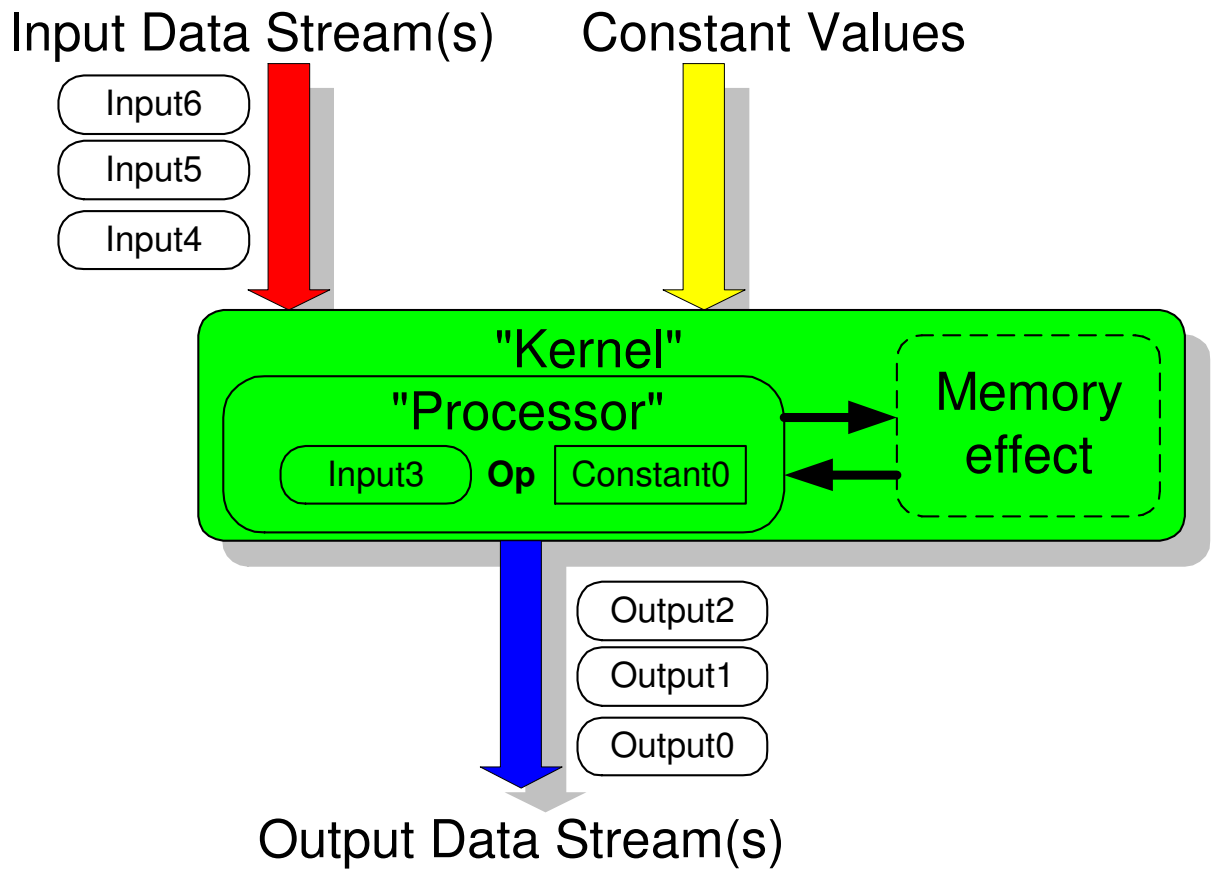
- GPU performs stream-based computing
  - Input texture images are inputted to GPU which calculates and outputs each pixel color data
- Performance of GPUs
  - Recently: GPU performance improves twice every 6 months
    CPU improves twice every 18 month (Moore's law: 18-24 months)
  - GeForce7 achieves 300GFLOPS
    Core2Duo achieves 8GFLOPS
- GPUs in nowadays:
  - Are programmable
  - Have very high performance floating point units
  - High data parallelism (color elements RGBA)

→ Can it become an high performance computing platform?

Organization of this research work



Flow-model


Input Data Stream(s)
Constant Values
Input6
Input5
Input4
"Kernel"
"Processor"
Input3
Op
Constant0
Memory effect
Output2
Output1
Output0
Output Data Stream(s)

Local machine execution

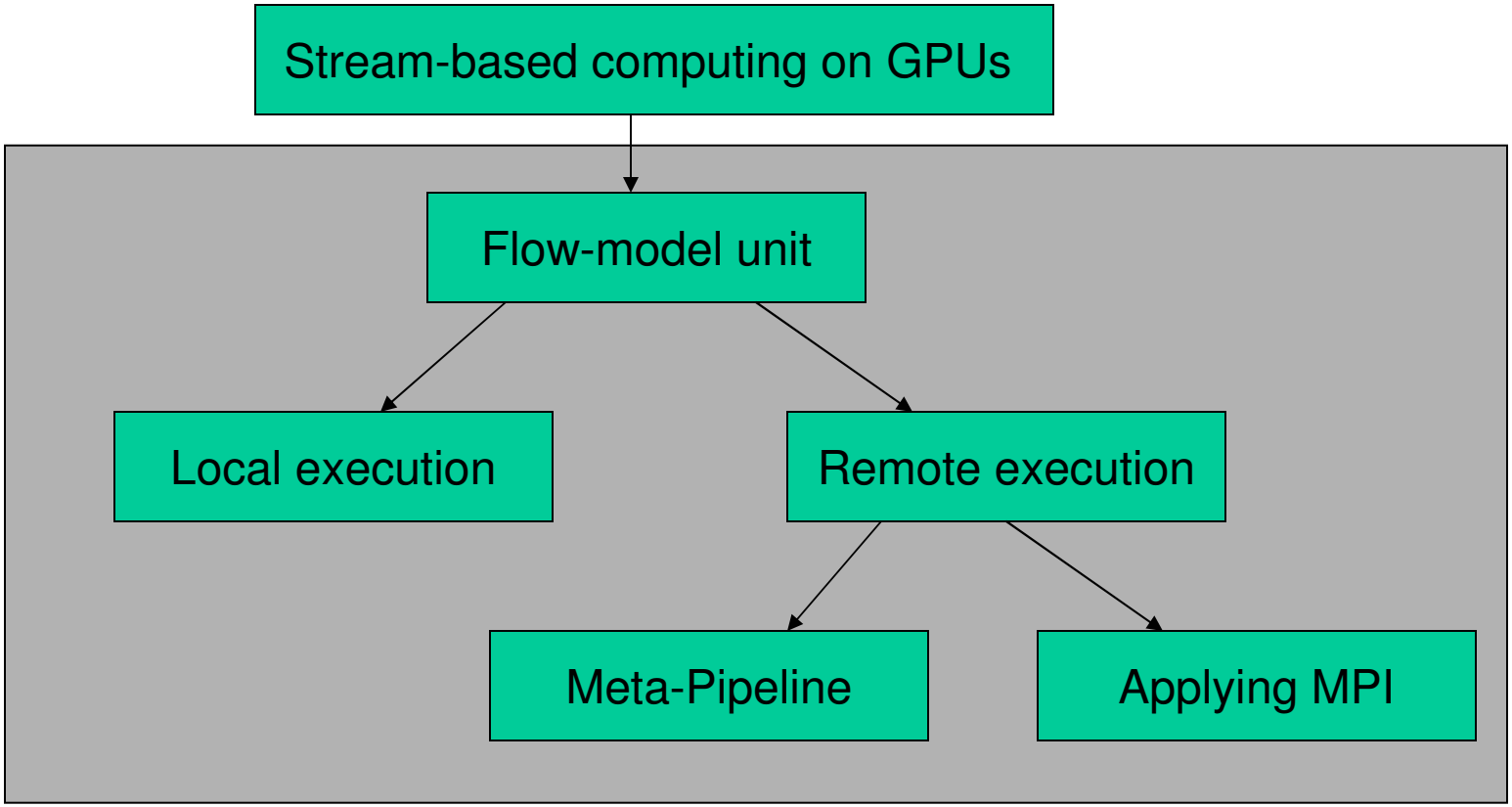
- Map the flow-model unit to a shader
  - Input data stream: Texture image input
  - Output data stream: Frame buffer
  - Program: pixel shader program

- What the application needs to do
  - Initialize input data stream
  - Execute flow-model unit
  - Read back output data stream

Caravela platform and tools

- Why this designation?

  Caravela is named after the speedy vessel commanded by Pedro Álvares Cabral, the Portuguese explorer who discovered Brazil in 1500


Stream-based computing on GPUs
Flow-model unit
Local execution
Remote execution
Meta-Pipeline
Applying MPI

Caravela platform

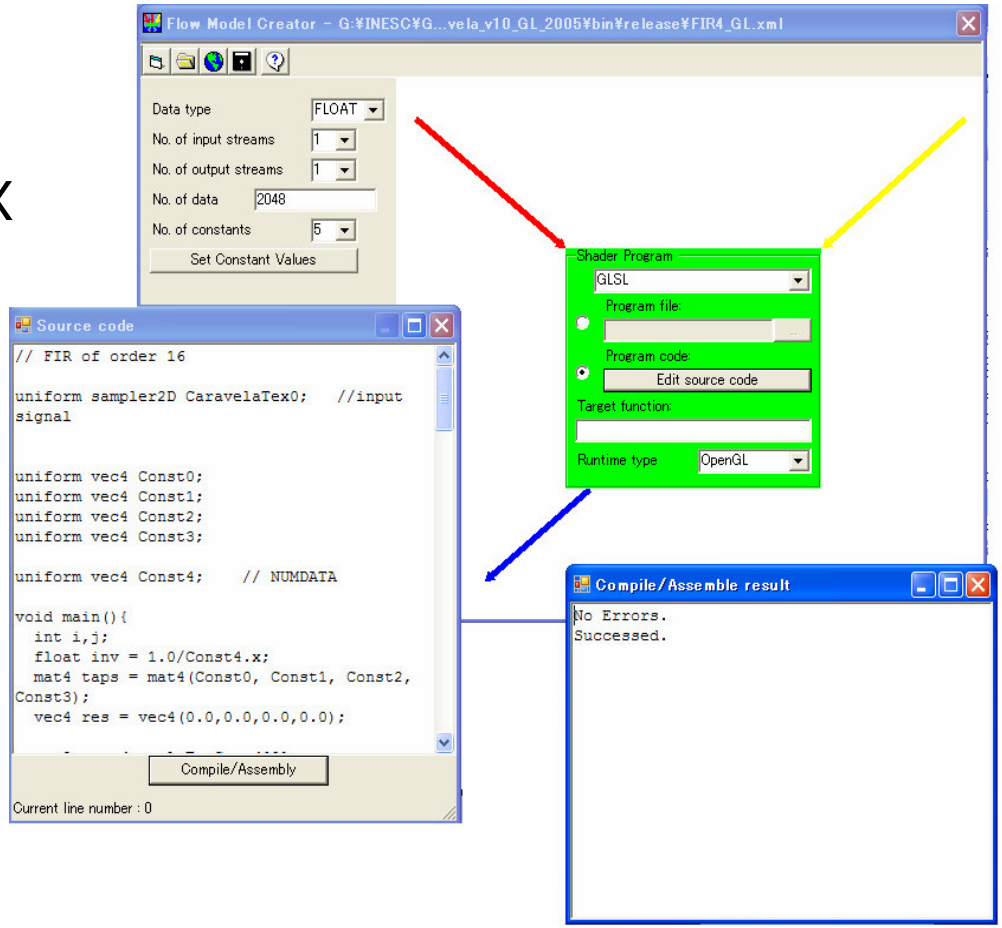
- Creation of the flow-model unit
- Caravela library (for Windows and Linux)
  - Local execution on GPU

    Optimized for recursive computing
  - Remote execution GPUs

    WebServices via Simple Object Access Protocol (SOAP)
  - Meta-pipelining and as MPI extension

Creating of a flow-model unit

- Flow-model is an autonomous unit
  - Pixel Shader program: DirectX (Assembly, HLSL) OpenGL (GLSL)
- FlowModelCreator
  - GUI to create flow-model unit
  - Saves flow-model to XML file
- Benefits
  - Reproducible
  - Fit to distributed environment



Caravela library

- Resource concept in Caravela library
  - Machine: a machine that includes adapters
  - Adapter: an video adapter that includes shaders
  - Shader: a pixel processor
- Programming steps
  1. Acquiring shader(s) in machine
  2. Creating flow-model units
  3. Mapping flow-model units to shader
  4. GetInputData
  5. Executing (firing) flow-model units
  6. GetOutputData

Shader

Adapter

Machine

Experimental setup

technology
from seed

inesc id lisboa

	Machine1	**Machine2**
Chipset	Nvidia nForce Ultra	Intel 945GM Express
Main Memory	AMD Opteron 170 @ 2GHz	Intel CoreDuo T2300 @ 1.66GHz
Graphics Board	MSI NX 7300GS 256MB DDR	nVidia GeForce Go 7400 128MB DDR2
Display Size	1280x1024	1280x800
OS	Windows XP Pro	Windows XP Home

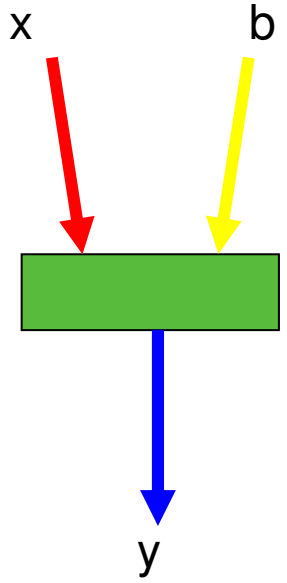
Application example
1D FIR filter with 16 taps

inesc id lisboa

OpenGL (GLSL)

```
void main(){
  int i,j;
  float inv = 1.0/Const4.x;
  vec4 res = vec4(0.0,0.0,0.0,0.0);

  vec2 coord = gl_TexCoord[0].xy;
  vec4 data0 = texture2D(CaravelaTex0, coord);
  coord.x+=inv;
  vec4 data1 = texture2D(CaravelaTex0, coord);
  coord.x+=inv;
  vec4 data2 = texture2D(CaravelaTex0, coord);

  // for x value
  for( j=0; j<4; j++ ){
     res.x += data0[j] * Const0[j];
     res.x += data1[j] * Const1[j];
  }

// for y value
  for(j=1;j<4;j++)
      res.y += data0[j] * Const0[j-1];
  res.y += data1[0] * Const0[3];
  for( j=1; j<4; j++ )
      res.y += data1[j] * Const1[j-1];
  res.y += data2[0] * Const1[3];

...
  gl_FragData[0] = res;
}
```

$$y_n = \sum_{i=0}^{15} b_i * x_{n-i}$$

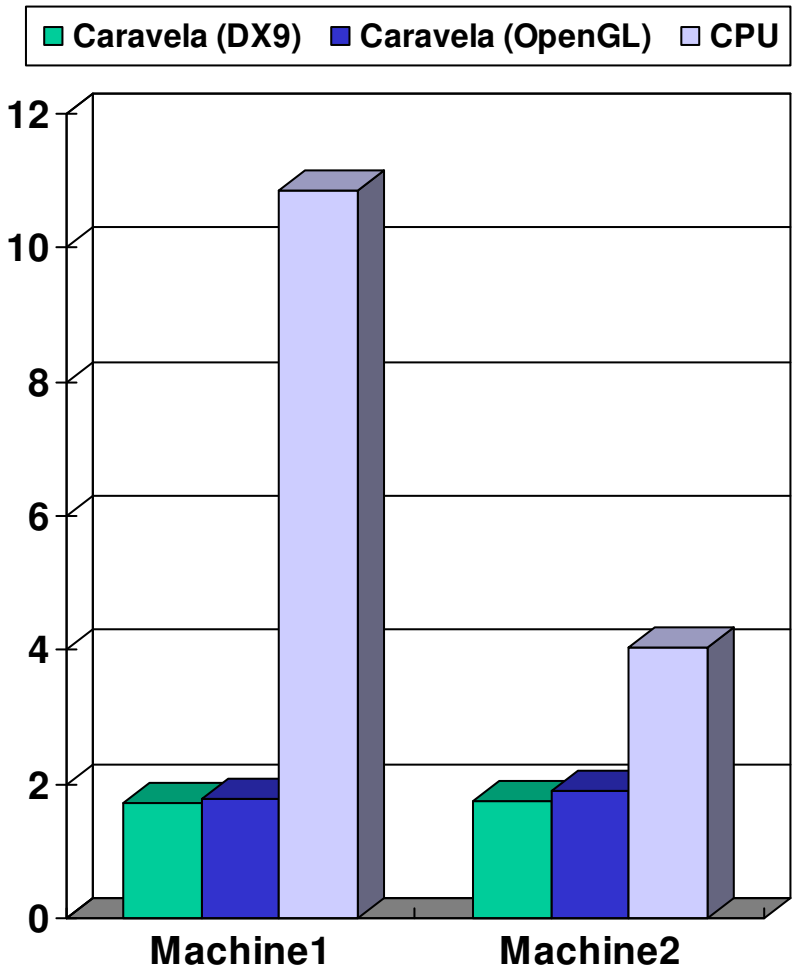


DirectX (HLSL)

```
void main( in float2 t0:  TEXCOORD0,
           out float4 oC0: COLOR0){
  int j;
  float inv = 1.0/Const4.x;
  float4 res = 0;
  float2 coord = t0;

  float4 data0 = tex2D(CaravelaTex0, coord);
  coord.x += inv;
  float4 data1 = tex2D(CaravelaTex0, coord);
  coord.x += inv;
  float4 data2 = tex2D(CaravelaTex0, coord);

 // for x value
   for( j=0; j<4; j++ )
      res.x += data0[j] * taps[j][0];
   for( j=0; j<4; j++ )
      res.x += data1[j] * taps[j][1];
// for y value
  for(j=1;j<4;j++)
      res.y += data0[j] * taps[j-1][0];
 res.y += data1[0] * taps[3][0];
 for( j=1; j<4; j++ )
    res.y += data1[j] * taps[j-1][1];
 res.y += data2[0] * taps[3][1];
 oC0 = res;
}
```

Result of FIR filter

- 1Mega samples for input matrix with 30 iterations
- 4-10 times faster than CPU-based execution



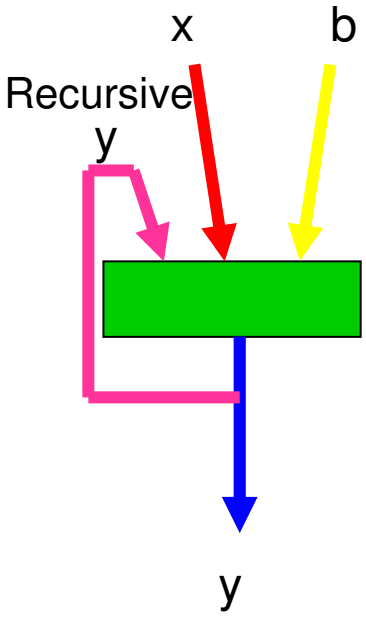
Application example
1D IIR filter with 16 taps

inesc id lisboa

$$y_n = \sum_{i=0}^{7} b_i * x_{n-i} + \sum_{k=1}^{8} b_k * y_{n-k}$$

OpenGL GLSL

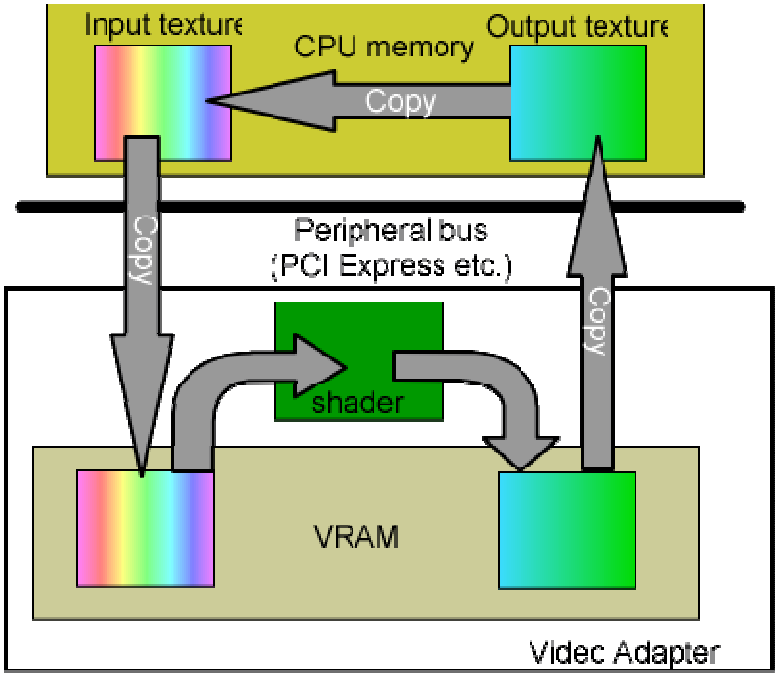
```
void main(){
  int i,j;
  float inv = 1.0/Const4.x;
  vec4 res = vec4(0.0,0.0,0.0,0.0);

  vec2 coord = gl_TexCoord[0].xy;
  vec4 data0 = texture2D(CaravelaTex0, coord);
  coord.x+=inv;
  vec4 data1 = texture2D(CaravelaTex0, coord);
  coord.x+=inv;
  vec4 data2 = texture2D(CaravelaTex0, coord);

  // for x value
  for( j=0; j<4; j++ ){
     res.x += data0[j] * Const0[j];
     res.x += data1[j] * Const1[j];
  }

// for y value
  for(j=1;j<4;j++)
      res.y += data0[j] * Const0[j-1];
  res.y += data1[0] * Const0[3];
  for( j=1; j<4; j++ )
      res.y += data1[j] * Const1[j-1];
  res.y += data2[0] * Const1[3];
...
  gl_FragData[0] = res;
}
```



DirectX HLSL

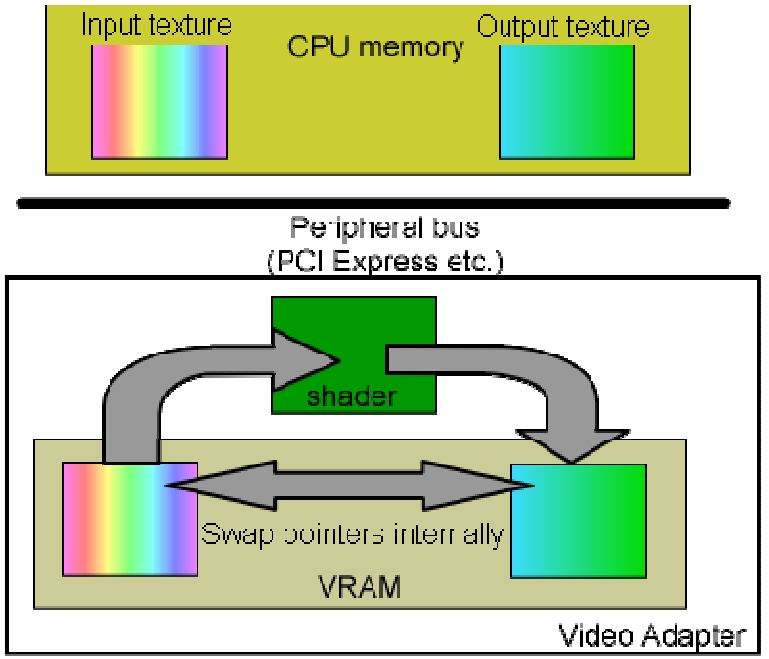
```
void main( in float2 t0:  TEXCOORD0,
           out float4 oC0: COLOR0){
  int j;
  float inv = 1.0/Const4.x;
  float4 res = 0;
  float2 coord = t0;
  float4 data0 = tex2D(CaravelaTex0, coord);
  coord.x += inv;
  float4 data1 = tex2D(CaravelaTex0, coord);
  coord.x += inv;
  float4 data2 = tex2D(CaravelaTex0, coord);
 // for x value
   for( j=0; j<4; j++ )
      res.x += data0[j] * taps[j][0];
   for( j=0; j<4; j++ )
      res.x += data1[j] * taps[j][1];
// for y value
  for(j=1;j<4;j++)
      res.y += data0[j] * taps[j-1][0];
  res.y += data1[0] * taps[3][0];
  for( j=1; j<4; j++ )
      res.y += data1[j] * taps[j-1][1];
   res.y += data2[0] * taps[3][1];
…
 oC0 = res;
  }
```

Memory effect for recursive computing



Copy method (DirectX)

copies VRAM data output to CPU memory and back to VRAM

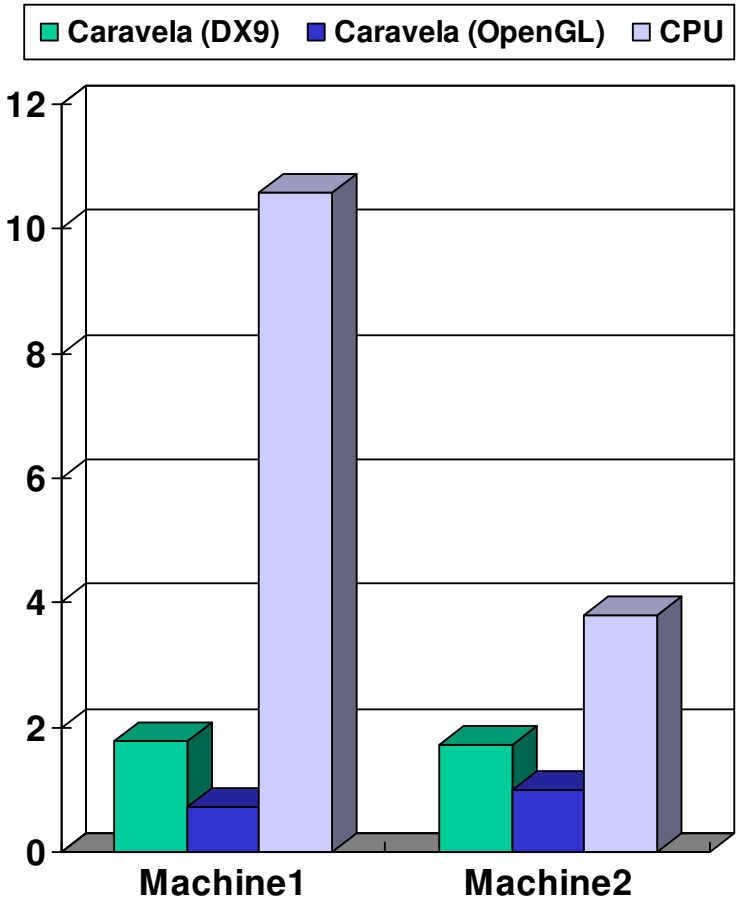


Swap method (OpenGL)

Exchanges pointers of input and output data in GPU side

Result of IIR Filter

- 1Mega samples for input matrix with 30 iterations
- 4-10 times faster than CPU-based execution
- The swap function on OpenGL reduces 50% regarding the DirectX implementation



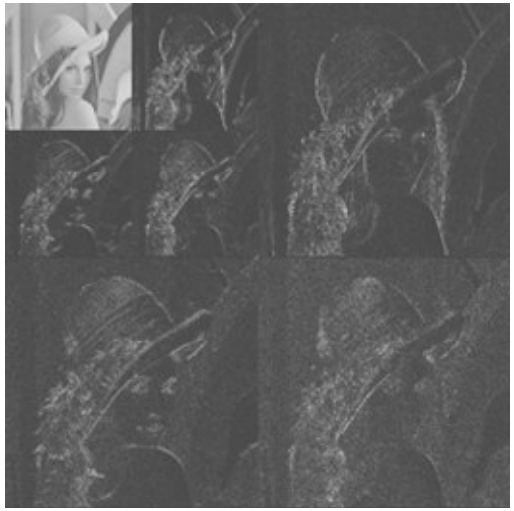
Application example
2D-DWT

- Example of a 2-level 2D-DWT decomposition

<table>
<tr><td>LL_2</td><td>HL_2</td><td rowspan="2">HL_1</td></tr>
<tr><td>LH_2</td><td>HH_2</td></tr>
<tr><td colspan="2">LH_1</td><td>HH_1</td></tr>
</table>

2D-DWT sub-bands

Input image



Output of 2D-DWT

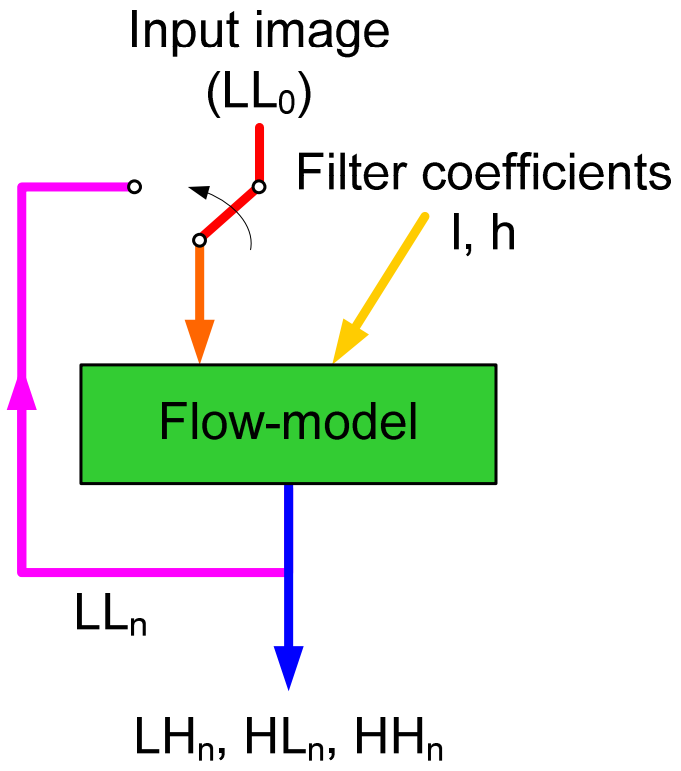
Application example
2D-DWT

- For each iteration:
  - LL, HL, LH and HH are computed
  - LL is sent back to the input (swap)
- Stops at the desired decomposition level
  - New problem: we are continuously decreasing data size
- Shader program equations:

$$LL_n(i,j) = \sum_{k=0}^{K}\sum_{m=0}^{M} LL_{n-1}(2i+k)(2j+m)l(k)l(m)$$

$$LH_n(i,j) = \sum_{k=0}^{K}\sum_{m=0}^{M} LL_{n-1}(2i+k)(2j+m)l(k)h(m)$$

…



Application example
LDPC decoder

- Low-Density Parity-Check codes are amoung the most powerful error correcting codes known (BER < 10^{-8})

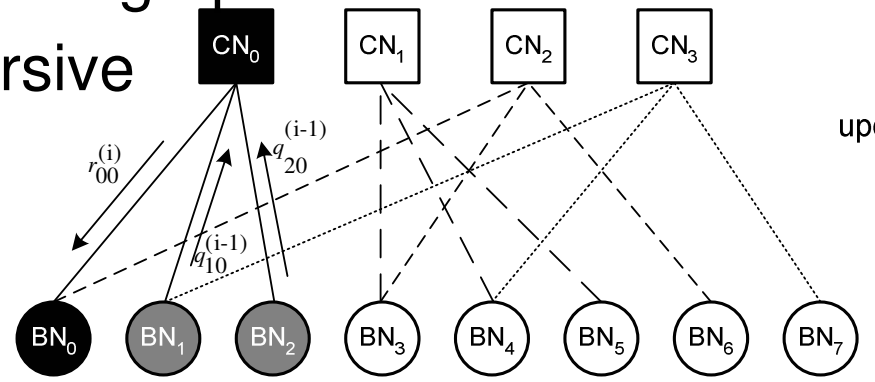
- LDPC decoders demand huge computational Power

H =

1	1	1	0	0	0	0	0
0	0	0	1	1	1	0	0
1	0	0	1	0	0	1	0
0	1	0	0	1	0	0	1

(8 bit nodes checked by 4 check node equations)

- Belief bropagation:Tanner graph and Sum Product recursive Algorithm (SPA)



Check node 0 updating bit node 0

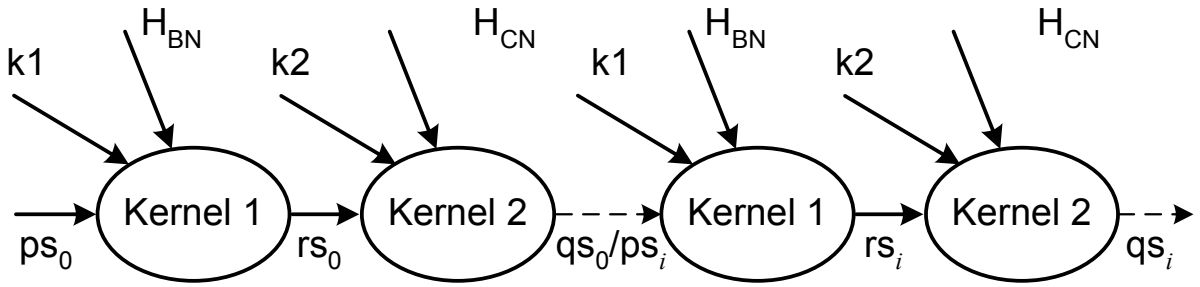
Application example
LDPC

- (Example: *Kernel 1*) Computes the message sent from CN_m to BN_n, that indicates the probability of BN_n (0 or 1)

$$r_{mn}^{(i)}(0) = \frac{1}{2} + \frac{1}{2} \prod_{n' \in N(m) \setminus n} (1 - 2q_{n'm}^{(i-1)}(1)) \qquad r_{mn}^{(i)}(1) = 1 - r_{mn}^{(i)}(0)$$

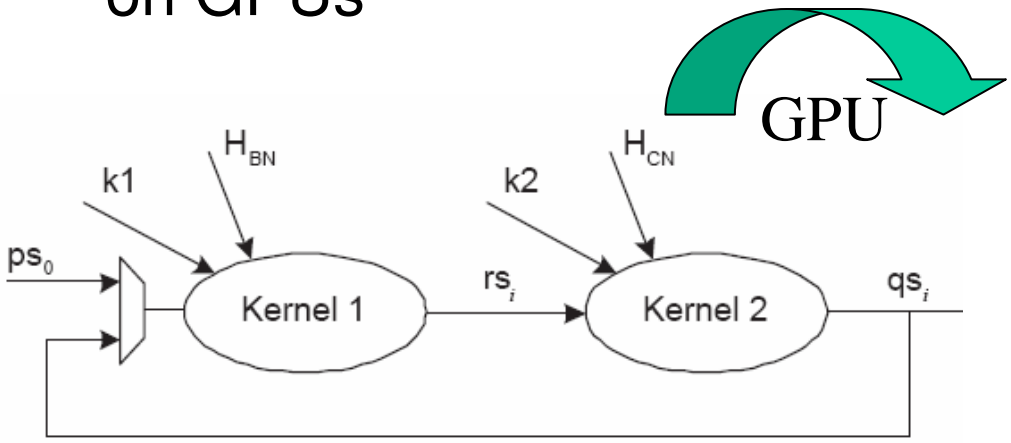
- SDF: kernels from stream-based LDPC decoding

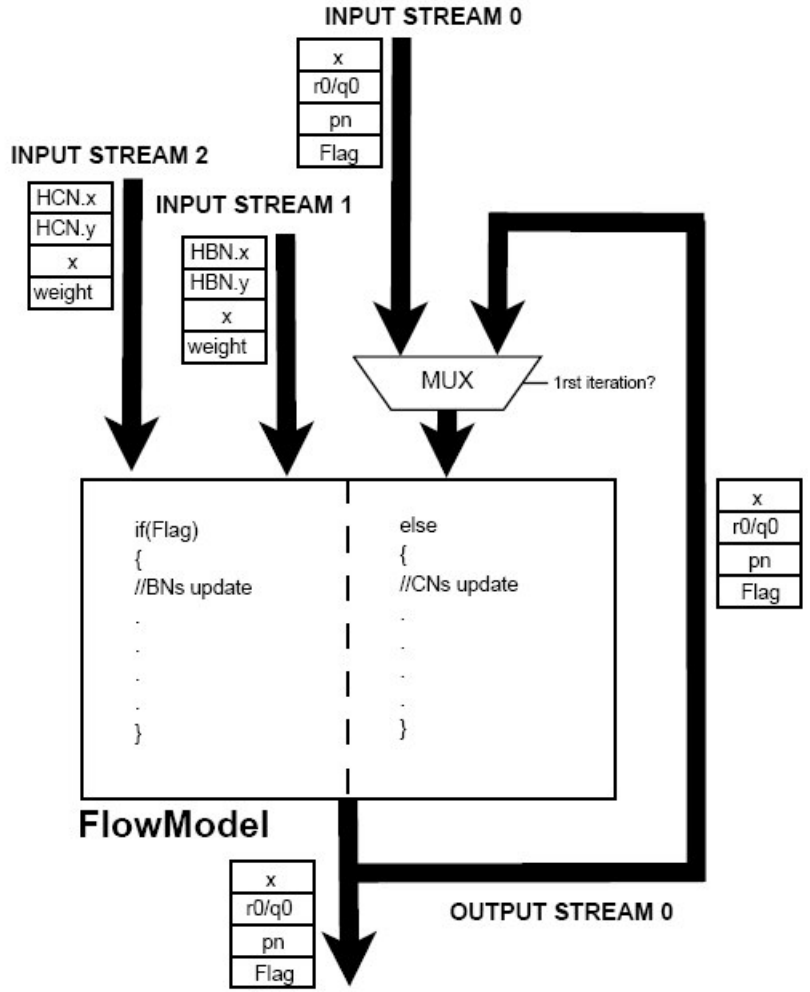
HBN, HCN: linear data structures representing **H** for streaming computing



Application example
LDPC

- Folded SDF for LDPC decodin on GPUs

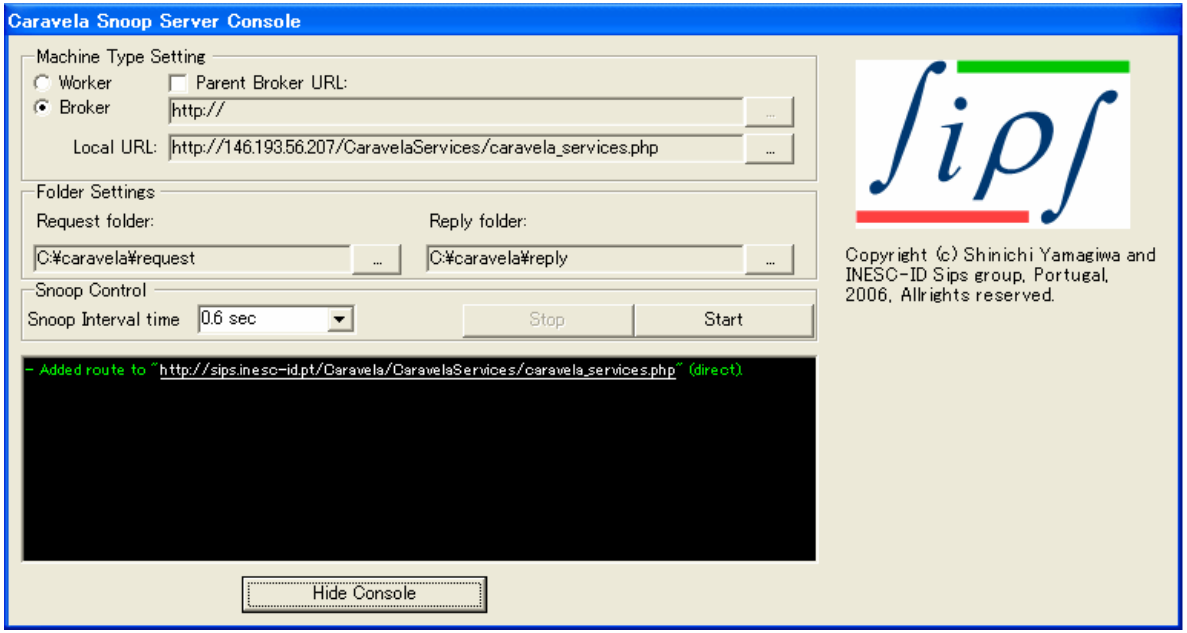




- Significant speedup!
  (Paper submitted with results for GPU and also to MMX)

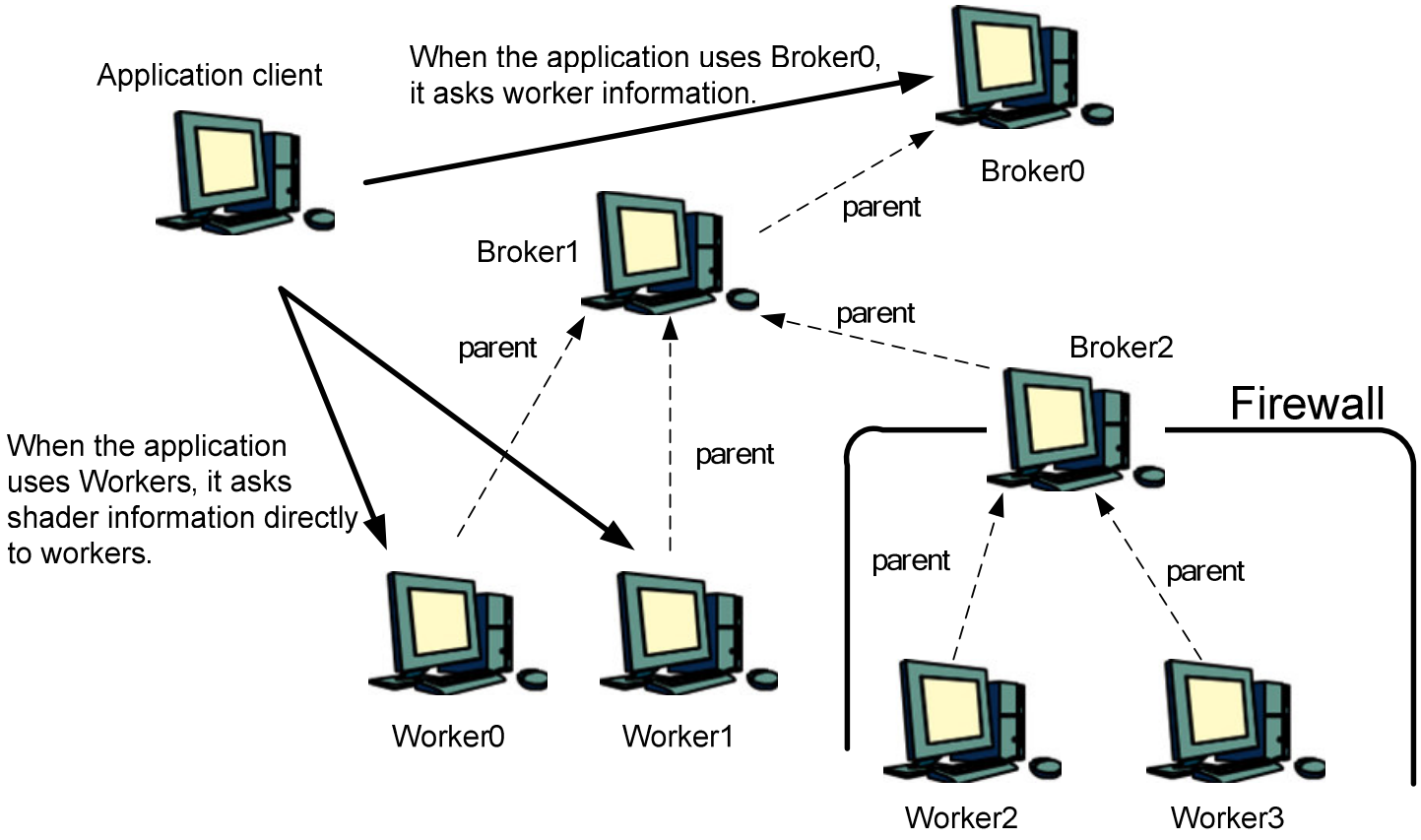
Remote execution

- Caravela Snoop Server
  - Creates a virtual network using WebServices
    - **Worker** : flow-model execution
    - **Broker** : maintains routing information to Workers
  - Caravela Library
    - Creates a remote machine
    - Queries shader processors via a broker machine.



Caravela network

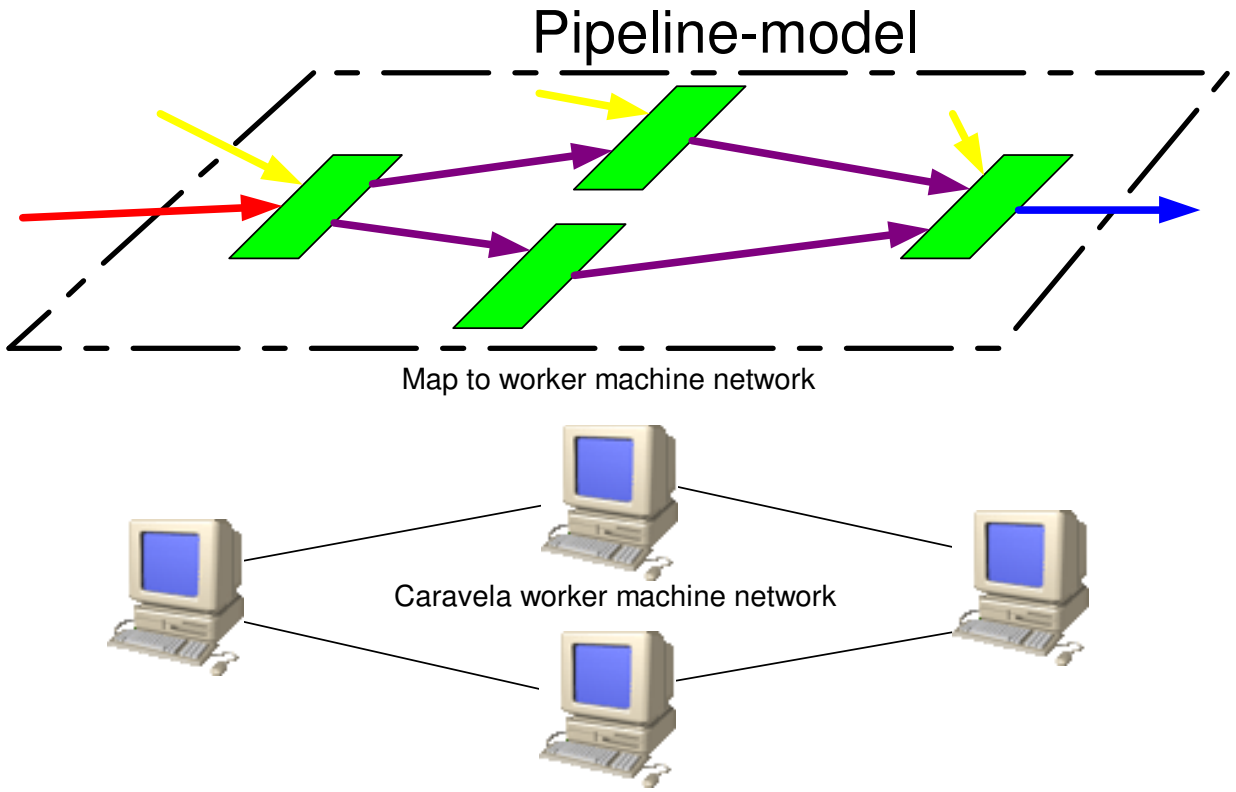
Machines in the Internet



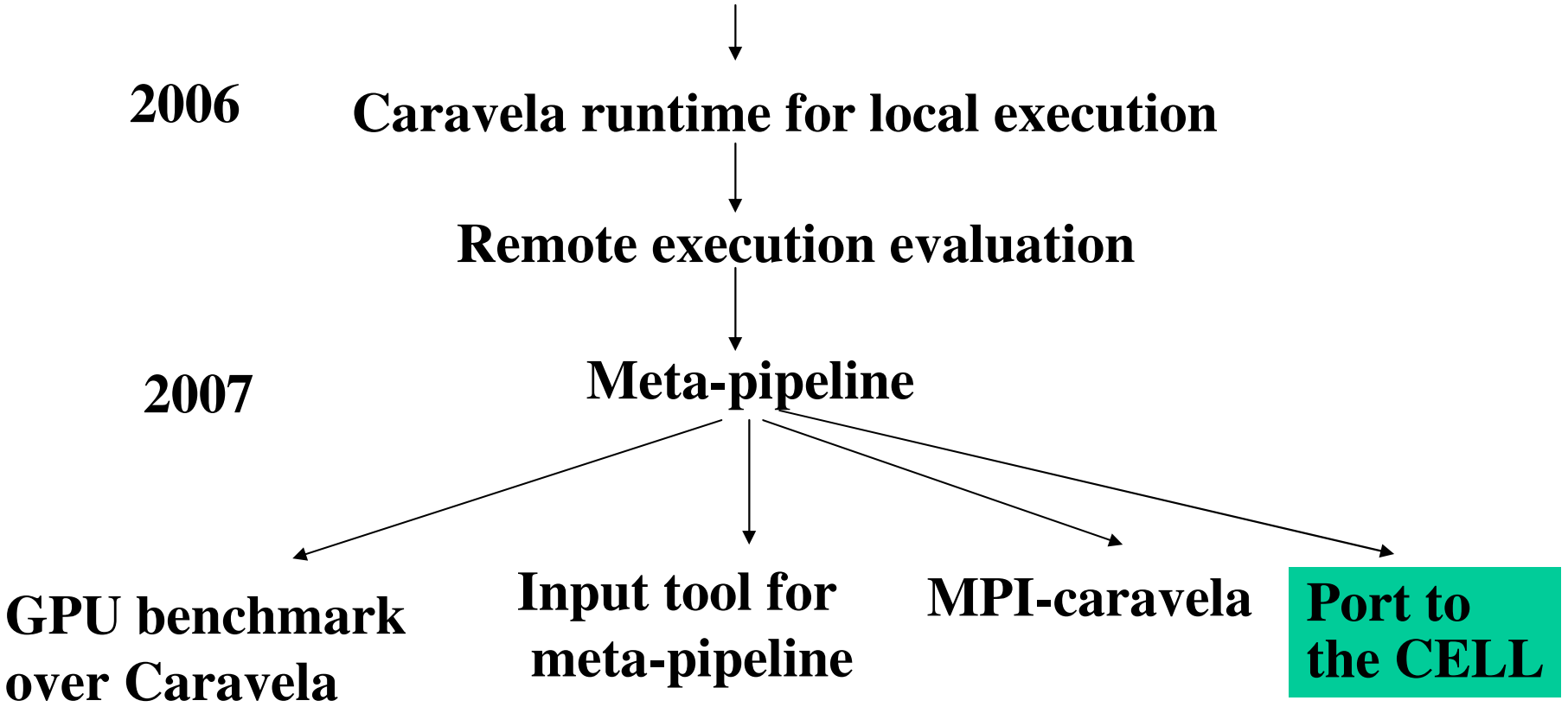
Machines in a LAN

Distributed computing: Meta-pipeline

- To create a processing pipeline with multiple flow-model units
- Flow-model units are virtually connected and mapped to distributed resources.



Roadmap for the Caravela project



Publications

1. Shinichi Yamagiwa, Leonel Sousa, "Caravela: A Novel Environment for stream-based distributed computing", IEEE Computer Magazine, May 2007, pp.76-83
2. Shinichi Yamagiwa, Leonel Sousa, "*Design and implementation of a stream-based distributed computing platform using graphics processing units*", ACM International Conference on Computing Frontier, May 2007
3. Shinichi Yamagiwa, Leonel Sousa, Diogo Antão, "*Data buffering optimization methods toward a uniform programming interface for GPU-based applications*", ACM International conference of Computing Frontier, May 2007
4. Shinichi Yamagiwa, Leonel Sousa, Tomas Brandao, "*Meta-Pipeline: A new execution mechanism for distributed pipeline processing*", 6th International Symposium on Parallel and Distributed Computing (ISPDC 2007), August 2007

   Shinichi Yamagiwa, Leonel Sousa, "Identifying execution deadlocks on pipeline processing", submitted to the Europar Conference 2007

- Patent (pending)
  - "*Program execution method applied to data streaming in distributed heterogeneous computing environment*", Portuguese national patent

Access the Caravela webpage now!

technology
from seed

inesc id lisboa

http://www.caravela-gpu.org

INSTITUTO SUPERIOR TÉCNICO


technology
from seed
inesc id
lisboa