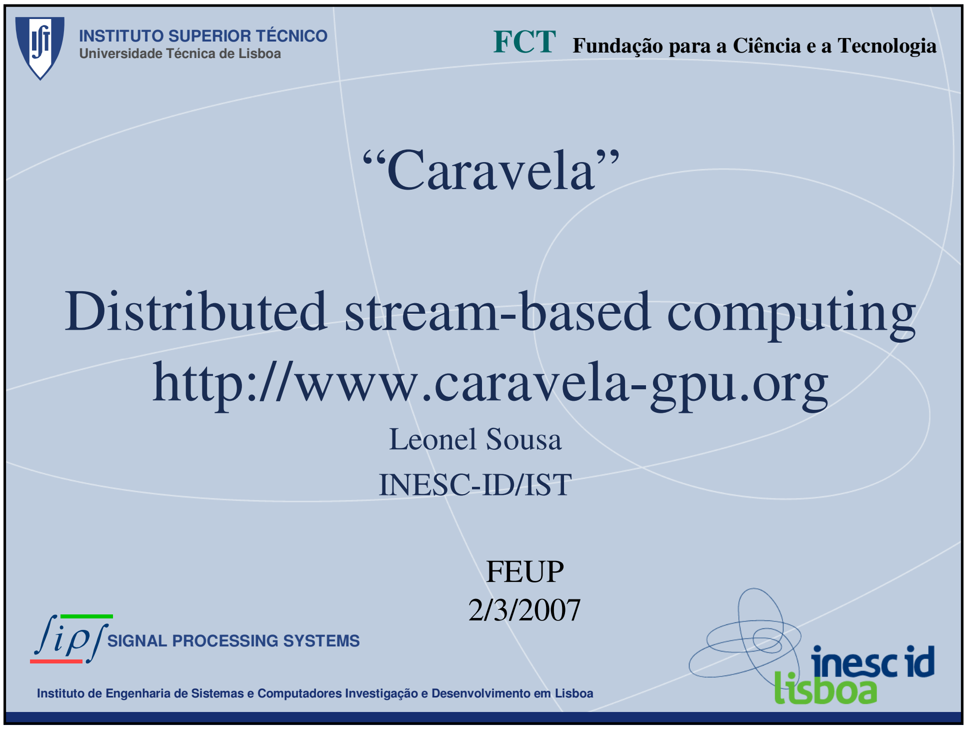


INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

FCT Fundação para a Ciência e a Tecnologia

"Caravela"

Distributed stream-based computing
http://www.caravela-gpu.org

Leonel Sousa
INESC-ID/IST

FEUP
2/3/2007

SIGNAL PROCESSING SYSTEMS

Instituto de Engenharia de Sistemas e Computadores Investigação e Desenvolvimento em Lisboa

inesc id lisboa

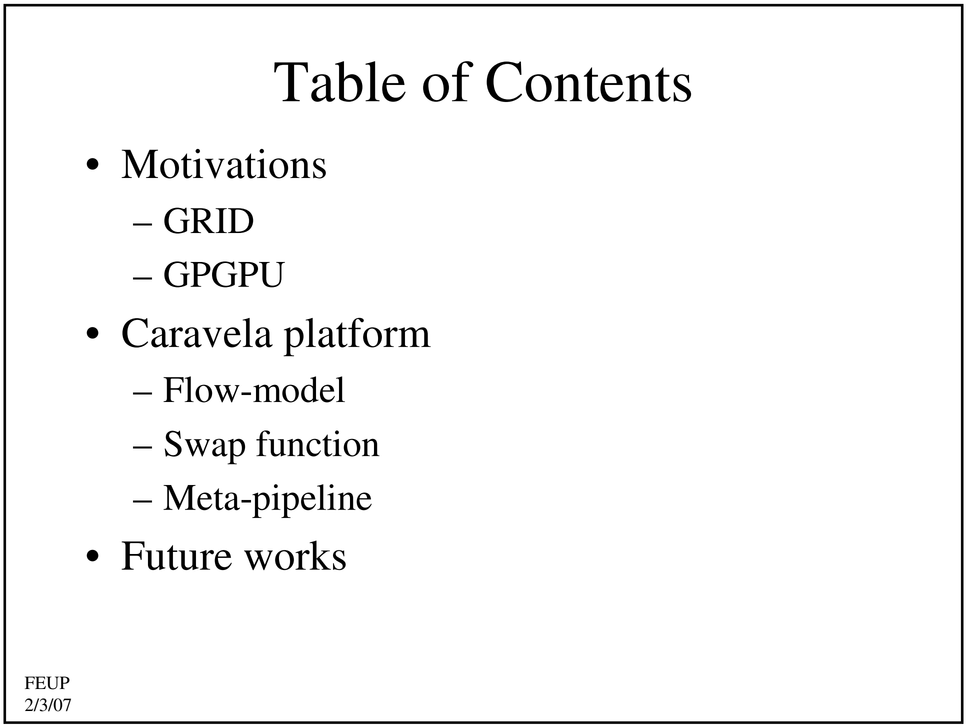


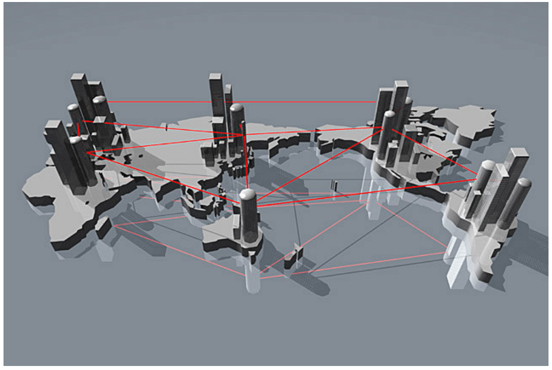
Table of Contents

- Motivations
  - GRID
  - GPGPU
- Caravela platform
  - Flow-model
  - Swap function
  - Meta-pipeline
- Future works

FEUP
2/3/07

GRID computing environment

- Distributed computing environment migrates
  - From supercomputers to cluster computers
  - Thereafter, to GRID that connects computing resources in the world
- Globus is popular tool for GRID



One of the most important issue is SECURITY

Security on GRID

- Users on GRID want:
  - Highly secure communication among processing units
  - Highly secure execution environment
- Contributors on GRID want:
  - Very safe execution environment with resource restrictions
- Current solutions
  - Account creation in each GRID machine
  - Restrictions for resources
  - Data encryption for communication

→ **These are not enough solutions.**

We need to address the security problems by applying a new execution mechanism

GPU

(General Purpose processing on Graphics Processing Unit)

- Performance of GPUs
  - Recently: GPU performance improvement achieves twice per 6 months CPU achieves twice in every 18 month according to Moore's law
  - GeForce7 achieves 300GFLOPS

    Core2Duo achieves 8GFLOPS
- Recent GPU is programmable
  - GPU has very high performance floating point units
  - Stream-based computation with color elements (i.e. RGBA) (high data parallelism)

GPGPU

(General Purpose processing on Graphics Processing Unit)

- Recent GPUS are programmable
  - Processing power can be exploited for general purpose processing

⇩

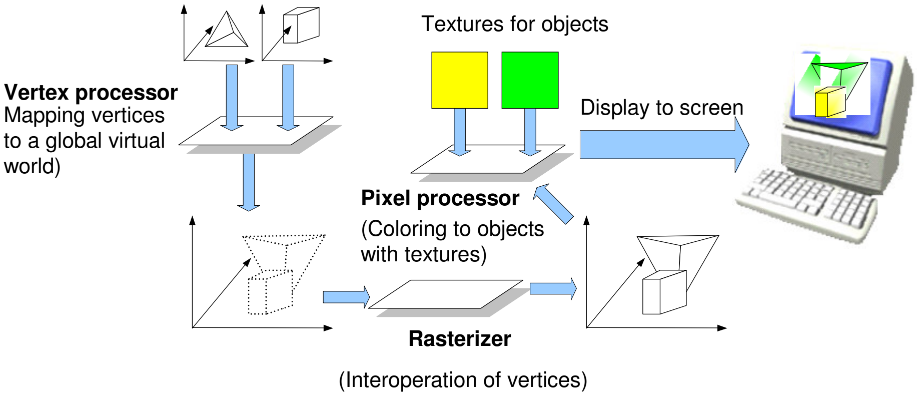
Apply general purpose applications to GPUs!

www.gpgpu.org

GPGPU

How to apply?
Texture mapping technique

Mechanism of Texture mapping



Vertex processor
Mapping vertices to a global virtual world)

Textures for objects

Display to screen

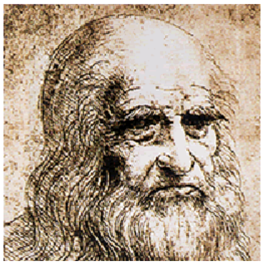
Pixel processor
(Coloring to objects with textures)

Rasterizer

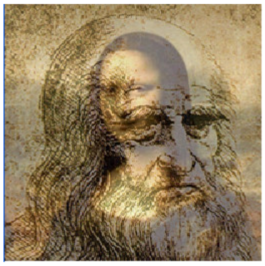
(Interoperation of vertices)

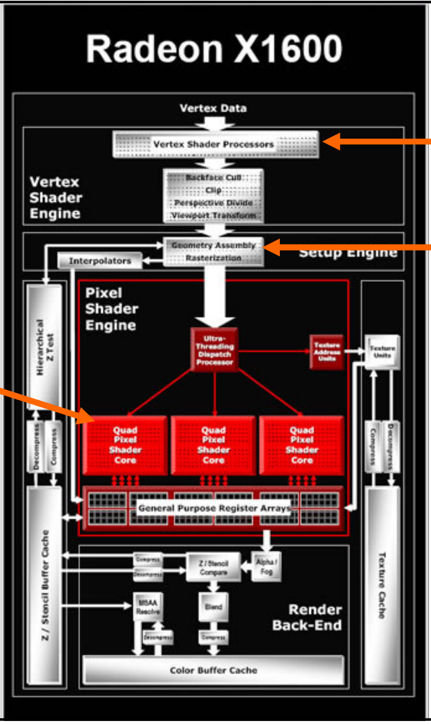
Demonstration of Texture mapping

- Alpha blending of two plane images



$$P' = Pa(1-\alpha)+Pb\alpha$$





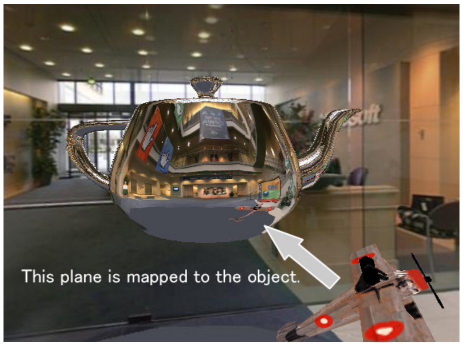
Vertex processor

Rasterizer

12 pixel processors

Environment mapping

- Making texture by using texture mapping technique
- Returns blended image to CPU memory



This process is the basic concept of GPGPU!

A new platform for GRID

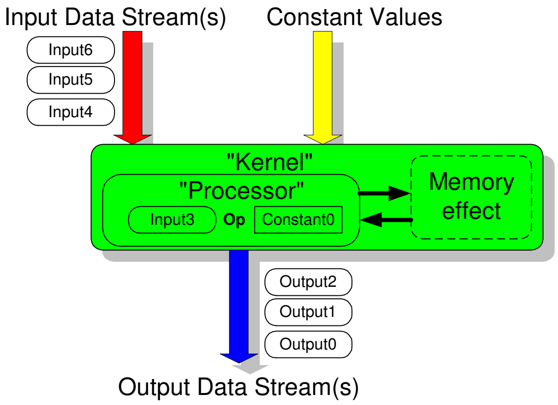
- Highly secured execution environment for GRID computing
- High performance execution of tasks

We propose the Caravela platform.

Caravela using GPU

- Stream-based computing using GPU
  - A new programming model called flow-model.
  - GPU is used as the processing unit
  - Flow-model will be mapped to GPUs
- Caravela Library
  - Provides an easy interface for programmer.
- Distributed flow-model management
  - Packs a flow-model into a data structure, and re-produces the kernel anywhere.
- Remote execution of flow-model
  - Provides a virtual network which executes flow-models in remote processing units

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)

- Flow-model includes;
  - Input and output data streams
  - A kernel program
  - Constant input values

Mapping flow-model into GPU

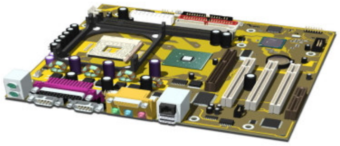
- Input data streams
  - Texture input
- Output data streams
  - Pixel processor’s output
- Kernel program
  - Shader program for pixel shader
- Constant input values
  - Constant of a shader program

Caravela Library

- Resource concept in Caravela library
  - Machine: a machine that includes adaptors
  - Adapter: an video adaptor that includes shaders
  - Shader: a pixel processor
- Programming steps
  1. Acquiring shaders
  2. Creating flow-models
  3. Mapping flow-models to shader
  4. Executing (firing) flow-model

Shader

Adapter


Machine

Caravela library functions

- CARAVELA_Initialize
  - Initialization of system
- CARAVELA_CreateMachine
  - Creation of a machine
- CARAVELA_QueryShader
  - Query for shader
- CARAVELA_CreateFlowModel
  - Creation of a flow-model data structure
- CARAVELA_SetShaderProgramToFlowModel
  - Assignment of a program to a flow-model
- CARAVELA_SetConstantsToFlowModel
  - Assignment of constant input values to a flow-model
- CARAVELA_MapFlowModelIntoShader
  - Mapping of a flow-model into a shader
- CARAVELA_GetInputData
  - Getting input data from shader
- CARAVELA_GetOutputData
  - Getting output data from shader
- CARAVELA_FireFlowModel
  - Firing flow-model

Distributed flow-model management

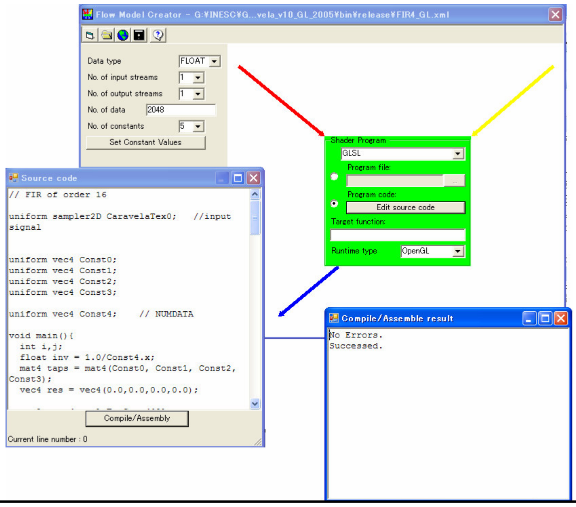
- Packing flow-model into XML file
  - Flow Model Creator
- Application re-produce flow-model by using Caravela Library
  - CARAVELA_CreateFlowModelFromFile
- The file can be accessed via HTTP
  - It can be saved in anywhere in the world.

Algorithms can be reused for applications
Thus, the productivity rises

FEUP
2/3/07

Flow-model Creator

- Helps to create a flow-model with GUI
- Saves flow-model to XML file



Flow Model Creator - G:¥INESC¥G...vela_v10_GL_2005¥bin¥release¥FIR4_GL.xml

Data type: FLOAT
No. of input streams: 1
No. of output streams: 1
No. of data: 2048
No. of constants: 5

Shader Program: GLSL
○ Program file:
● Program code:

Target function:
Runtime type: OpenGL

Source code

```
// FIR of order 16

uniform sampler2D CaravelaTex0;   //input
signal


uniform vec4 Const0;
uniform vec4 Const1;
uniform vec4 Const2;
uniform vec4 Const3;

uniform vec4 Const4;    // NUMDATA

void main(){
  int i,j;
  float inv = 1.0/Const4.x;
  mat4 taps = mat4(Const0, Const1, Const2,
Const3);
  vec4 res = vec4(0.0,0.0,0.0,0.0);
```

Current line number : 0

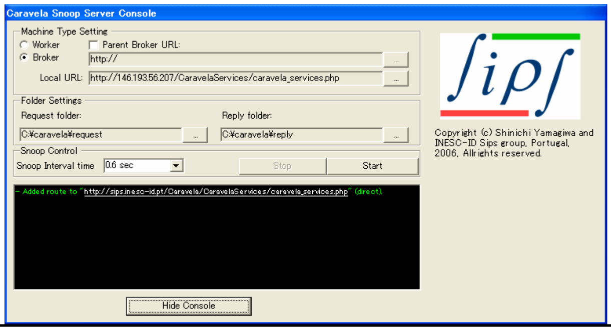
Compile/Assemble result

```
No Errors.
Successed.
```

FEUP
2/3/07

Remote execution of flow-model

- Caravela Snoop Server
  - Creates a virtual network using WebServices
    - Worker : execution of flow-model
    - Broker : maintains routing information for Worker
  - Support in Caravela Library
    - CARAVELA_CreateMachine(REMOTE_MACHINE)
    - CARAVELA_GetRemoteMachines


Caravela Snoop Server Console

Machine Type Setting
○ Worker
● Broker
□ Parent Broker URL: http://
Local URL: http://146.193.56.207/CaravelaServices/caravela_services.php

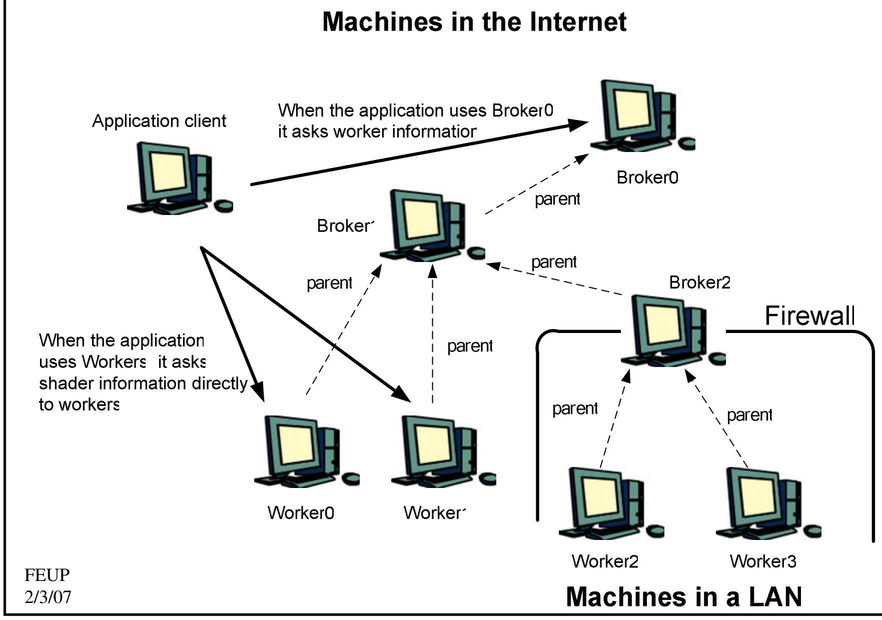
Folder Settings
Request folder: C:¥caravela¥request
Reply folder: C:¥caravela¥reply

Snoop Control
Snoop Interval time 0.6 sec

```
- Added route to "http://sips.inesc-id.pt/Caravela/CaravelaServices/caravela_services.php" (direct).
```

Copyright (c) Shinichi Yamagiwa and INESC-ID Sips group, Portugal, 2006, Allrights reserved.

Caravela network


Machines in the Internet

Application client

When the application uses Broker0 it asks worker informatior

Broker0

Broker1

Broker2

Firewall

parent

parent

parent

parent

parent

parent

When the application uses Workers it asks shader information directly to workers

Worker0

Worker1

Worker2

Worker3

Machines in a LAN

Local execution example

- Non-recursive application : FIR filter
- Recursive application : IIR filter

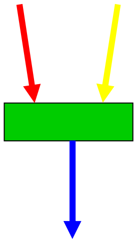
	Machine1	Machine2
Chipset	nForce4 Ultra	945GM Express
CPU	AMD Opteron 170 @2GHz	Intel CoreDuo T2300@1.66GHz
CPU memory	2x1GB DDR 400	2x512MB DDR2 533
Graphics	MSI NX7300GS 256MB DDR	nVIDIA GeForce Go 7400 128MB DDR2
OS	WindowsXP pro	WindowsXP home

FEUP
2/3/07

Local execution example : FIR filter

1D FIR filter with 16 taps:

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



DirectX

```
// FIR of order 16
sampler CaravelaTex0;   //input signal
float4x4 taps;
float4 Const4;    // NUMDATA

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);
 coord.x += inv;
  float4 data3 = 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];
…
```

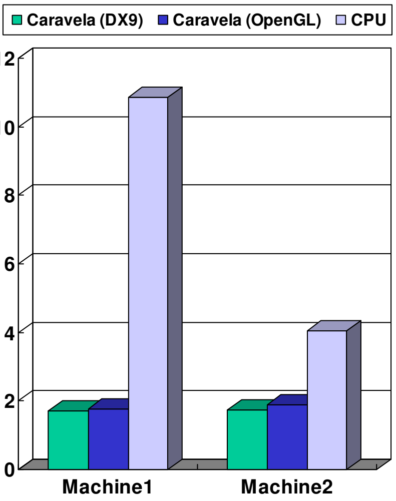
OpenGL

```
uniform sampler2D CaravelaTex0;
 //input signal
uniform vec4 Const0;
uniform vec4 Const1;
uniform vec4 Const2;
uniform vec4 Const3;
uniform vec4 Const4;    // NUMDATA
void main(){
  int i,j;
  float inv = 1.0/Const4.x;
  mat4 taps = mat4(Const0, Const1, Const2, Const3);
  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);
  mat4 data = mat4( data0, data1, data2, res);
…
```

FEUP
2/3/07

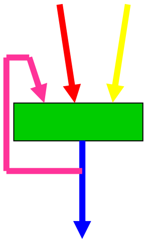
Result of FIR Filter

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



Local execution example : IIR filter

1D IIR filter with 16 taps: $y_n = \sum_{i=0}^{7} b_i * x_{n-i} + \sum_{k=1}^{8} b_k * y_{n-k}$



DirectX

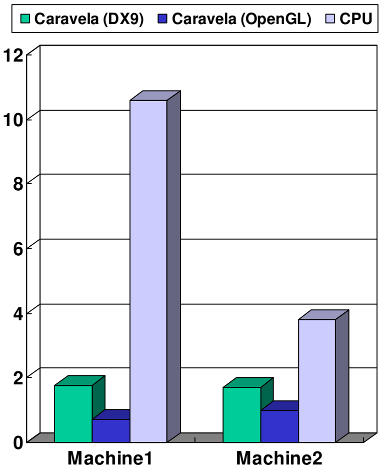
```
// IIR of order 8 for forward and 8 for feedback
sampler CaravelaTex0;   //input signal
sampler CaravelaTex1;   //previus output signal
float4x4 taps;
float4 Const4;    // NUMDATA
void main( in float2 t0:  TEXCOORD0,
          out float4 oC0: COLOR0
){
  int j;
  float inv = 1.0/Const4.x;
  float4 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];
…
```

OpenGL

```
uniform sampler2D CaravelaTex0;
 //input signal
uniform vec4 Const0;
uniform vec4 Const1;
uniform vec4 Const2;
uniform vec4 Const3;
uniform vec4 Const4;    // NUMDATA
void main(){
  int i,j;
  float inv = 1.0/Const4.x;
  mat4 taps = mat4(Const0, Const1, Const2, Const3);
  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);
  mat4 data = mat4( data0, data1, data2, res);
…
```

Result of IIR Filter

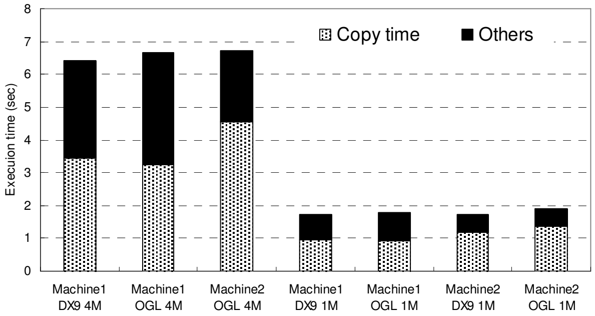
- 1Mega samples for input matrix with 30 iterations
- 4-10 times faster than CPU-based execution



FEUP
2/3/07

Considering copy operations

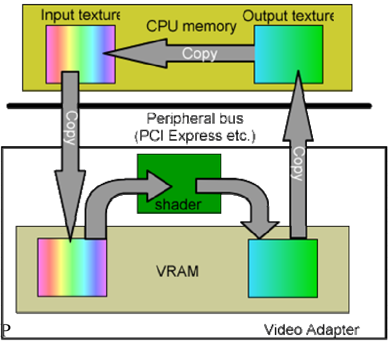
- 50-70% of execution time is copy operations to feedback the output data to input data.



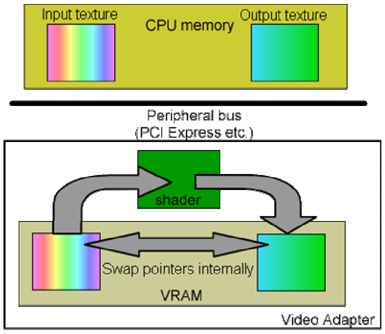
FEUP
2/3/07

Swap frame function

- Reduction of overhead caused by recursive application
- Copy method (DirectX, OpenGL)
  - copies output data to input stream among CPU memory and VRAM
- Swap pointer method (OpenGL)
  - Exchanges pointers of input and output data in CPU side
- Swap frame method (OpenGL)
  - Exchanges pointers of input and output data in GPU side



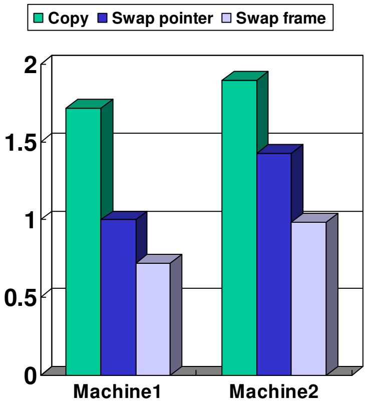
Copy method (DirectX)



Swap method (OpenGL)

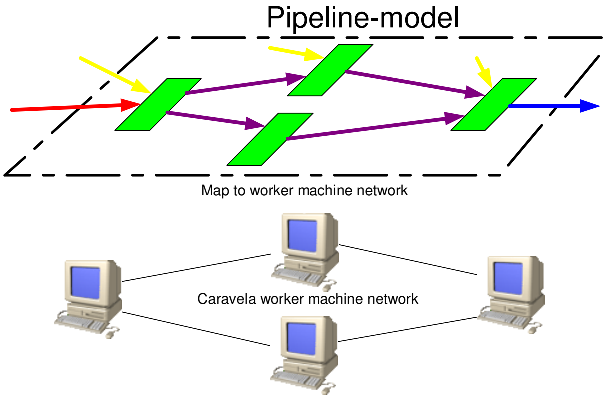
Optimization by swap function

- Swap frame method achieves twice faster performance than copy method.
- Caravela library provides a swap frame method automatically.



Future works

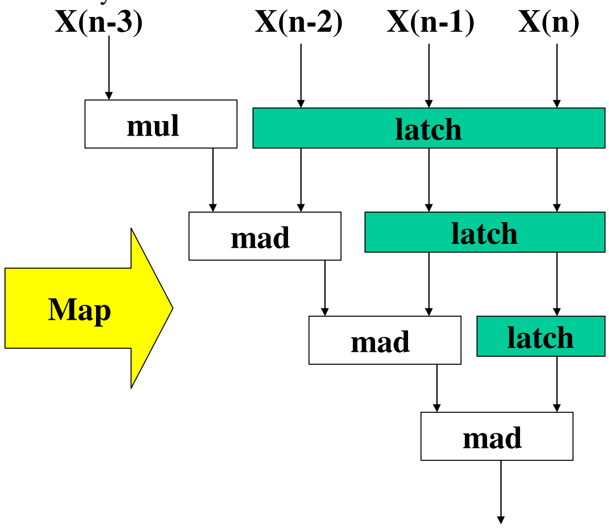
- Meta-pipeline
  - Creating a processing pipeline with multiple flow-models
  - Flow-models are connected virtually, and mapped to distributed resources.


Pipeline-model
Map to worker machine network
Caravela worker machine network

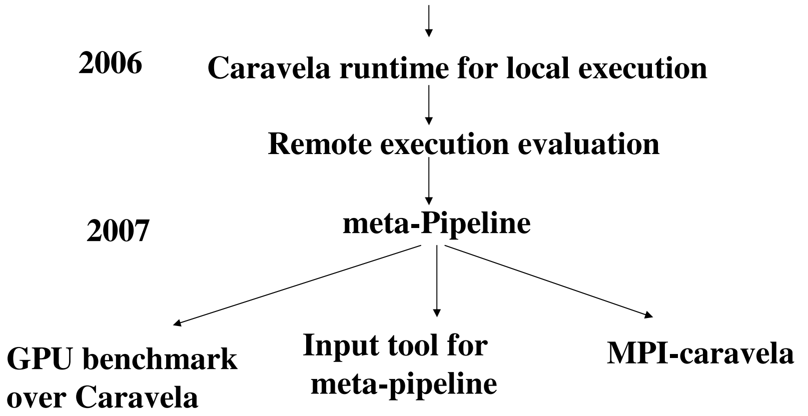
Toward the meta-pipeline…

- Hardware compiler
  - dumps flow-models to HDL.
  - Simulation is performed by Caravela.

```
ps_2_0

dcl_2d s0
dcl_2d s1
dcl_2d s2
dcl_2d s3

dcl t0.xy
dcl t1.xy
dcl t2.xy
dcl t3.xy

texld r0, t0, s0
texld r1, t1, s1
texld r2, t2, s2
texld r3, t3, s3

mul r4, r0,c0
mad r4, r1, c1, r4
mad r4, r2, c2, r4
mad r4, r3, c3, r4

mov oC0, r4
```



Roadmap for the Caravela project



2006 Caravela runtime for local execution

Remote execution evaluation

2007 meta-Pipeline

GPU benchmark over Caravela

Input tool for meta-pipeline

MPI-caravela

Publications

- Papers
  - Accepted
    - Shinichi Yamagiwa, Leonel Sousa, "*Caravela: A Novel Environment for stream-based distributed computing*", IEEE Computer Magazine.
    - Shinichi Yamagiwa, Leonel Sousa, "*Design and implementation of a stream-based distributed computing platform using graphics processing units*", ACM International Conference on Computing Frontier 2007.
    - Shinichi Yamagiwa, Leonel Sousa, Diogo Antão, "Data buffering optimization methods toward a uniformed programming interface for GPU-based applications", ACM International conference of Computing Frontier 2007.
  - Submitted
    - Shinichi Yamagiwa, Leonel Sousa, "Identifying execution deadlocks on pipeline processing", Europar Conference 2007.
- Patent (pending)
  - "*Program execution method applied to data streaming in distributed heterogeneous computing environment*", Portuguese national patent.

Access to Caravela webpage now!

http://www.caravela-gpu.org