Parallel Image Processing with CUDA

– A case study with the Canny Edge Detection Filter –

Daniel Weingaertner

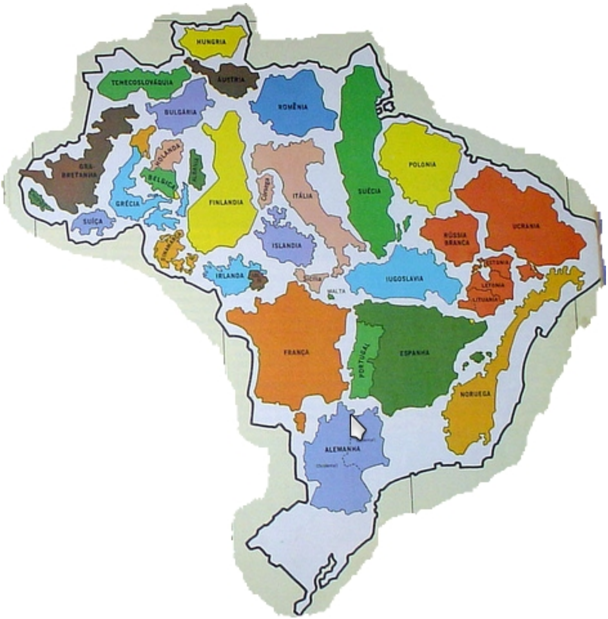
Informatics Department
Federal University of Paraná - Brazil

Hochschule Regensburg
02.05.2011

Summary

Paraná – Brazil

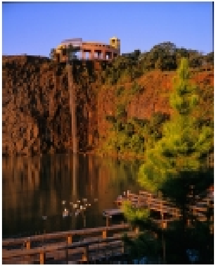
Brazil – Europe



Paraná


MATO GROSSO DO SUL
SÃO PAULO
PARANÁ
PARAGUAI
ARGENTINA
SANTA CATARINA
OCEANO ATLANTICO
SERRA DO MAR
0 75 150 225 300 mi
0 150 300 450 km
50°
25°
Paranapanema
Paraná
Ivaí
Tibají
Ribeira
Iguaçu
Piquiri
Cornélio Procópio
Londrina
Maringá
Arapongas
Apucarana
Cianorte
Umuarama
Campo Mourão
Guaíra
Ortigueira
Curiúva
Ivaiporã
Telêmaco Borba
Mal. Cândido Rondon
Toledo
Cascavel
Pitanga
Castro
Ponta Grossa
Pq. Nac. do Iguaçu
Laranjeiras do Sul
Guarapuava
Irati
Campo Largo
Curitiba
Paranaguá
Foz do Iguaçu
Dois Vizinhos
São Mateus do Sul
Quitandinha
Francisco Beltrão
Pato Branco
Baía de Paranaguá

Curitiba



Federal University of Paraná

Informatic Department

Informatics Department

Undergraduate: Bachelor in Computer Science

- 8 semesters course
- 80 incoming students per year

Bachelor in Biomedical Informatics

- 8 semesters course
- 30 incoming students per year

Graduate: Master and PhD in Computer Science

- Algorithms, Image Processing, Computer Vision, Artificial Intelligence
- Databases, Scientific Computing and Open Source Software, Computer-Human Interface
- Computer Networks, Embedded Systems

Summary

Insight Toolkit (ITK)

- Created in 1999, Open Source, Multi platform, Object Oriented (Templates), Good documentation and support

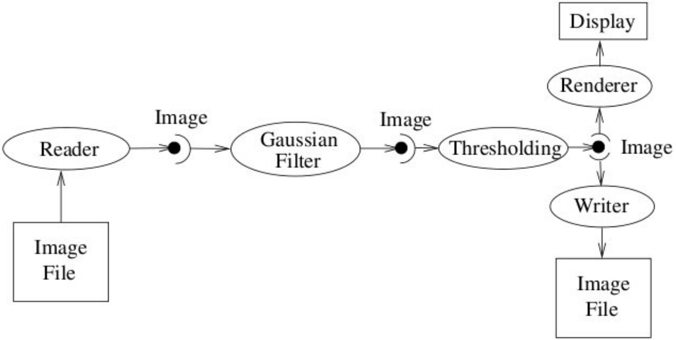


Figure: Image Processing Workflow in ITK

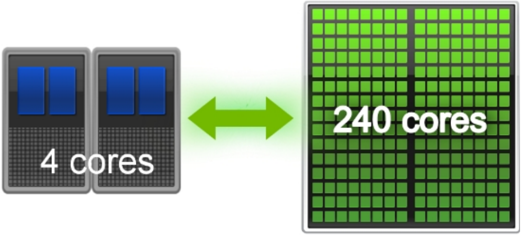
ITK - Sample code

```cpp
#include "itkImage.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkCannyEdgeDetectionImageFilter.h"

typedef itk::Image<float,2>                      ImageType;
typedef itk::ImageFileReader< ImageType > ReaderType;
typedef itk::ImageFileWriter< ImageType > WriterType;
typedef itk::CannyEdgeDetectionImageFilter< ImageType, ImageType > CannyFilter;

int main (int argc, char** argv){

  ReaderType::Pointer reader = ReaderType::New();
  reader->SetFileName( argv[1] );
  reader->Update();

  CannyFilter::Pointer canny = CannyFilter::New();
  canny->SetInput(reader->GetOutput());
  canny->SetVariance( atof( argv[3] ) );
  canny->SetUpperThreshold( atoi( argv[4] ) );
  canny->SetLowerThreshold( atoi( argv[5] ) );
  canny->Update();

  WriterType::Pointer writer = WriterType::New();
  writer->SetFileName( argv[2] );
  writer->SetInput( canny->GetOutput() );
  writer->Update();

  return EXIT_SUCCESS;
}
```

Summary

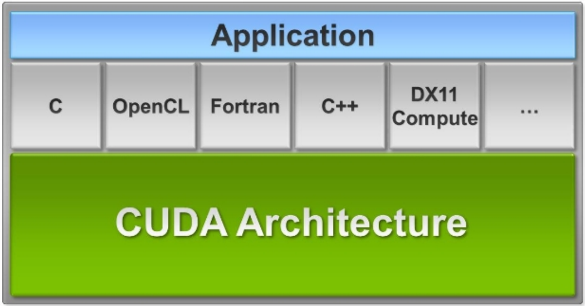
What is GPGPU Computing?

- The use of the GPU for general purpose computation
- CPU and GPU can be used concurrently
- To the end user, its simply a way to run applications faster.

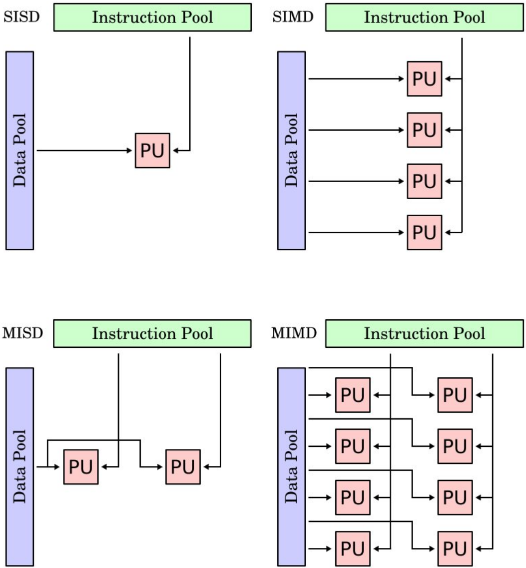


What is CUDA?

- CUDA = *Compute Unified Device Architecture*.
- *General-Purpose Parallel Computing Architecture*.
- Provides libraries, C language extension and hardware driver.

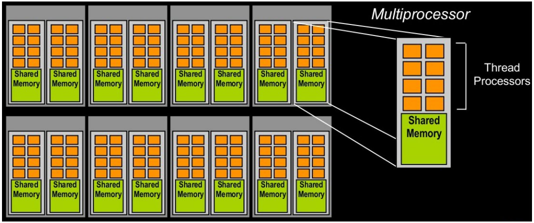


Parallel Processing Models

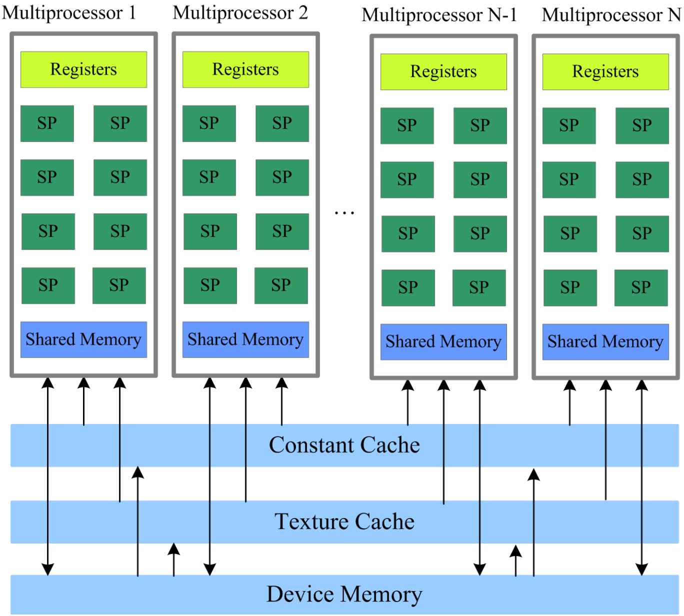


Single-Instruction Multiple-Thread Unit

- Creates, handles, schedules and executes groups of 32 threads (*warp*).
- All threads in a *warp* start at the same point.
- But they are "free" to jump to different code positions independently.



CUDA Architecture Overview



Optimization Strategies for CUDA

Main optimization strategies for CUDA involve:

- Optimized/careful memory access
- Maximization of processor utilization
- Maximization of non-serialized instructions

CUDA - Sample Code

```c
#include <stdio.h>
#include <assert.h>
#include <cuda.h>
void incrementArrayOnHost(float *a, int N)
{
  int i;
  for (i=0; i < N; i++) a[i] = a[i]+1.f;
}
__global__ void incrementArrayOnDevice(float *a, int N)
{
  int idx = blockIdx.x*blockDim.x + threadIdx.x;
  if (idx<N) a[idx] = a[idx]+1.f;
}
int main(void)
{
  float *a_h, *b_h;           // pointers to host memory
  float *a_d;                 // pointer to device memory
  int i, N = 10000;
  size_t size = N*sizeof(float);
  a_h = (float *)malloc(size);
  b_h = (float *)malloc(size);
  cudaMalloc((void **) &a_d, size);
  for (i=0; i<N; i++) a_h[i] = (float)i;
  cudaMemcpy(a_d, a_h, sizeof(float)*N, cudaMemcpyHostToDevice);
  incrementArrayOnHost(a_h, N);
  int blockSize = 256;
  int nBlocks = N/blockSize + (N%blockSize == 0?0:1);
  incrementArrayOnDevice <<< nBlocks, blockSize >>> (a_d, N);
  cudaMemcpy(b_h, a_d, sizeof(float)*N, cudaMemcpyDeviceToHost);
  free(a_h); free(b_h); cudaFree(a_d);
}
```

Summary

Integrating CUDA Filters into ITK Workflow

ITK community suggests:

- Re-implement filters where parallelizing provides significant speedup
- Consider the entire workflow: copying to/from the GPU is very time consuming

Careful!

"Premature optimization is the root of all evil!" (Donald Knuth)

Integrating CUDA Filters into ITK Workflow

ITK community suggests:

- Re-implement filters where parallelizing provides significant speedup
- Consider the entire workflow: copying to/from the GPU is very time consuming

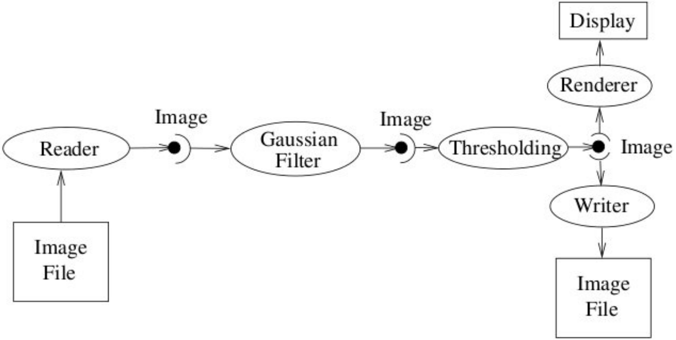
Careful!

"Premature optimization is the root of all evil!" (Donald Knuth)

CUDA Insight Toolkit (CITK)

Changes to ITK

- Slight architecture change: CudaImportImageContainer
- Backwards compatible
- Data transfer between HOST and DEVICE only "on demand"
- Allows for filter chaining inside the DEVICE



Summary

CudaCanny

- itkCudaCannyEdgeDetectionImageFilter

Algorithm 1 Canny Edge Detection Filter

Gaussian Smoothing
Gradient Computation
Non-Maximum Supression
Histeresis

Gradient Computation with Sobel Filter

- itkCudaSobelEdgeDetectionImageFilter

-1	0	1
-2	0	2
-1	0	1

(a) Sobel X

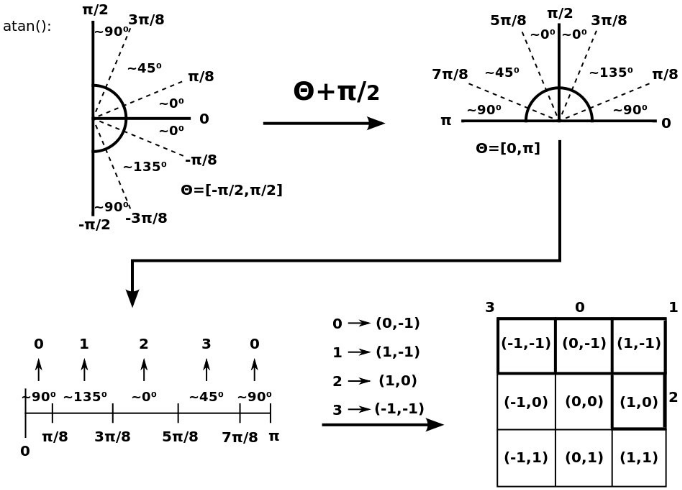
1	2	1
0	0	0
-1	-2	-1

(b) Sobel Y

$$L_v = \sqrt{L_x^2 + L_y^2} \tag{1}$$

$$\theta = \arctan\left(\frac{L_y}{L_x}\right) \tag{2}$$

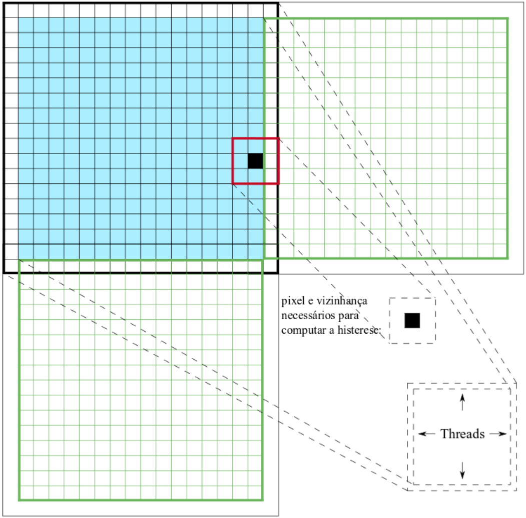
Optimization for Edge Direction Computation



Code Extract from CudaSobel

```
if ((pos.x) && ((size.x-1)-pos.x) && (pos.y) && ((size.y-1)-pos.y)){

  diagonal.x = tex1Dfetch(texRef,(pixIdx-size.x-1));
  diagonal.y = tex1Dfetch(texRef,(pixIdx-size.x+1));
  diagonal.z = tex1Dfetch(texRef,(pixIdx+size.x-1));
  diagonal.w = tex1Dfetch(texRef,(pixIdx+size.x+1));
  cross.x = tex1Dfetch(texRef,(pixIdx-size.x));
  cross.y = tex1Dfetch(texRef,(pixIdx+size.x));
  cross.z = tex1Dfetch(texRef,(pixIdx-1));
  cross.w = tex1Dfetch(texRef,(pixIdx+1));

  /// SobelX
  g_i.x -= (diagonal.x+cross.z+cross.z+diagonal.z);
  g_i.x += (diagonal.y+cross.w+cross.w+diagonal.w);

  /// SobelY
  g_i.y -= (diagonal.z+cross.y+cross.y+diagonal.w);
  g_i.y += (diagonal.x+cross.x+cross.x+diagonal.y);
}

Magnitude[pixIdx] = sqrtf((g_i.x*g_i.x) + (g_i.y*g_i.y));

theta = (g_i.x != 0)*(int)(atanf(__fdividef(g_i.y,g_i.x))*__fdividef(180,M_PI)) + 90;
if (theta > 157) theta -= 158;
theta = ceilf(__fdividef(theta-22,45));
Direction[pixIdx] = make_short2(1-(theta == 0)-((theta == 1)<<1),(theta == 2)-1);

}
```

Histeresis Operation



Histeresis Algorithm

Algorithm 2 Histeresis on CPU

Transfers the Gradient/NMS images to the GPU
repeat
    Run the histeresis kernel on GPU
until no pixel changes status
Return edge image

Histeresis Algorithm

Algorithm 3 Histeresis on GPU

```
Load an image region with size 18x18 into shared memory
modified ← false
repeat
   modified_region ← false
   Synchronize threads of same multiprocessor
   if Pixel changes status then
      modified ← true
      modified_region ← true
   end if
   Synchronize threads of same multiprocessor
until modified_region = false
if modified = true then
   Update modified status on HOST
end if
```

Summary

Metodology

Hardware:

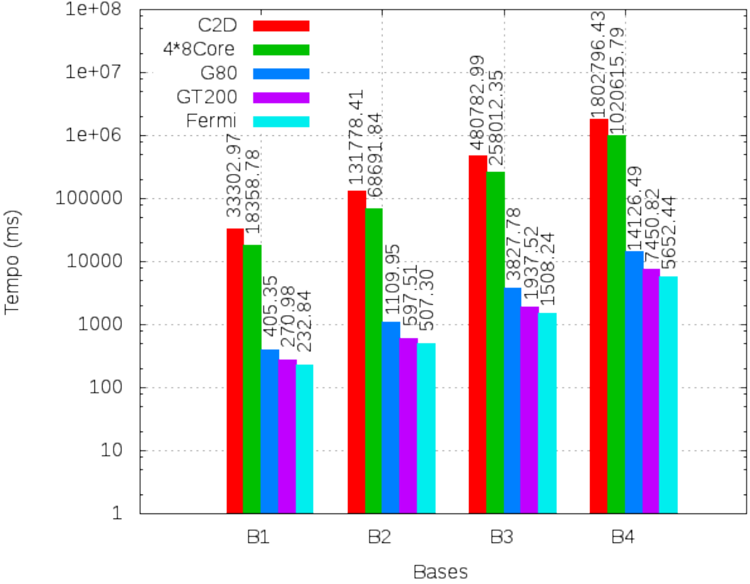
- Server:
  - CPU: 4x AMD Opteron(tm) Processor 6136 2,4GHz with 8 cores, each with 512 KB cache and 126GB RAM
  - GPU1: NVidia Tesla C2050 with 448 1,15GHz cores and 3GB RAM.
  - GPU2: NVidia Tesla C1060 com 240 1,3GHz cores and 4GB RAM.
- Desktop:
  - CPU: Intel®Core(TM)2 Duo E7400 2,80GHz with 3072 KB cache and 2GB RAM
  - GPU: NVidia GeForce 8800 GT with 112 1,5GHz cores and 512MB RAM.

Metodology

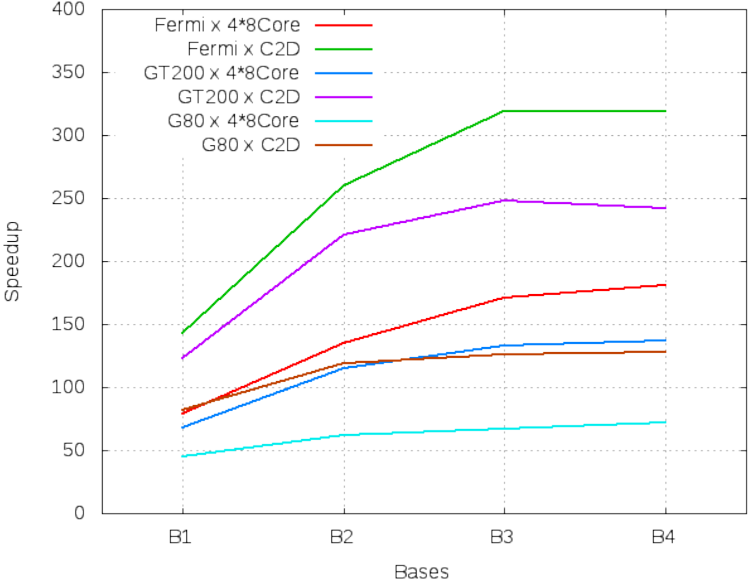
- Images from the Berkeley Segmentation Dataset

Base	Image resolution	Num. of Images
B1	321×481 e 481×321	100
B2	642×962 e 962×642	100
B3	1284×1924 e 1924×1284	100
B4	2568×3848 e 3848×2568	100

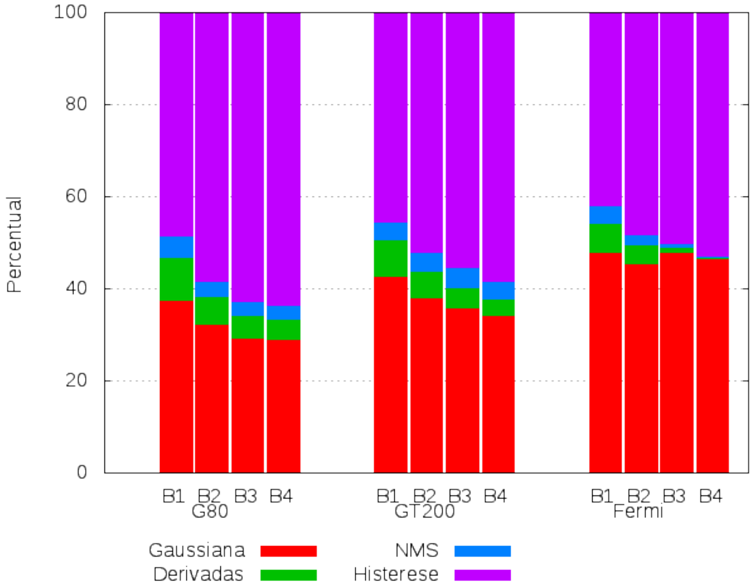
Performance Tests



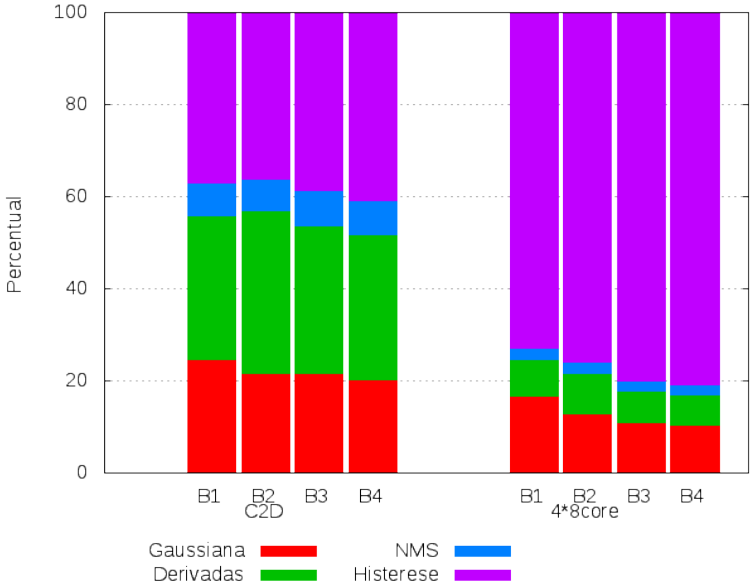
Performance Tests



Performance Tests



Performance Tests



Summary

Conclusion

Parallel Programming

- Parallel programming is definitely the way to go.
- Implement efficient parallel code is demanding.
- Programmer should know more details about the hardware, especially memory architecture.

Canny Filter with CUDA

- We had a great speedup on the edge detection filter
- Also noticed that the existing implementation is not efficient
- There is still a LOT of work if we want to parallelize ITK.

Conclusion

Parallel Programming

- Parallel programming is definitely the way to go.
- Implement efficient parallel code is demanding.
- Programmer should know more details about the hardware, especially memory architecture.

Canny Filter with CUDA

- We had a great speedup on the edge detection filter
- Also noticed that the existing implementation is not efficient
- There is still a LOT of work if we want to parallelize ITK.

Contact

Thank You!

Daniel Weingaertner
danielw@inf.ufpr.br