Processamento de áudio em tempo real utilizando dispositivos não convencionais:

Processamento paralelo com Pure Data e GPU.

André Jucovsky Bianchi
ajb@ime.usp.br

Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo

1 de outubro de 2011

Objetivos da apresentação

- Apresentar o ambiente Pure Data para processamento de sinais em tempo real.
- Apresentar a plataforma GPU para processamento paralelo de propósito geral.
- Motivar a utilização de processamento paralelo utilizando Pure Data e GPU e apresentar alguns trabalhos em andamento.

Estrutura da apresentação

Processamento em tempo real utilizando dispositivos não convencionais

Pure Data (Pd)

GPU

Pd + GPU

Próximos passos

Estrutura da apresentação

Processamento em tempo real utilizando dispositivos não convencionais

Pure Data (Pd)

GPU

Pd + GPU

Próximos passos

Processamento de áudio em tempo real utilizando dispositivos não convencionais

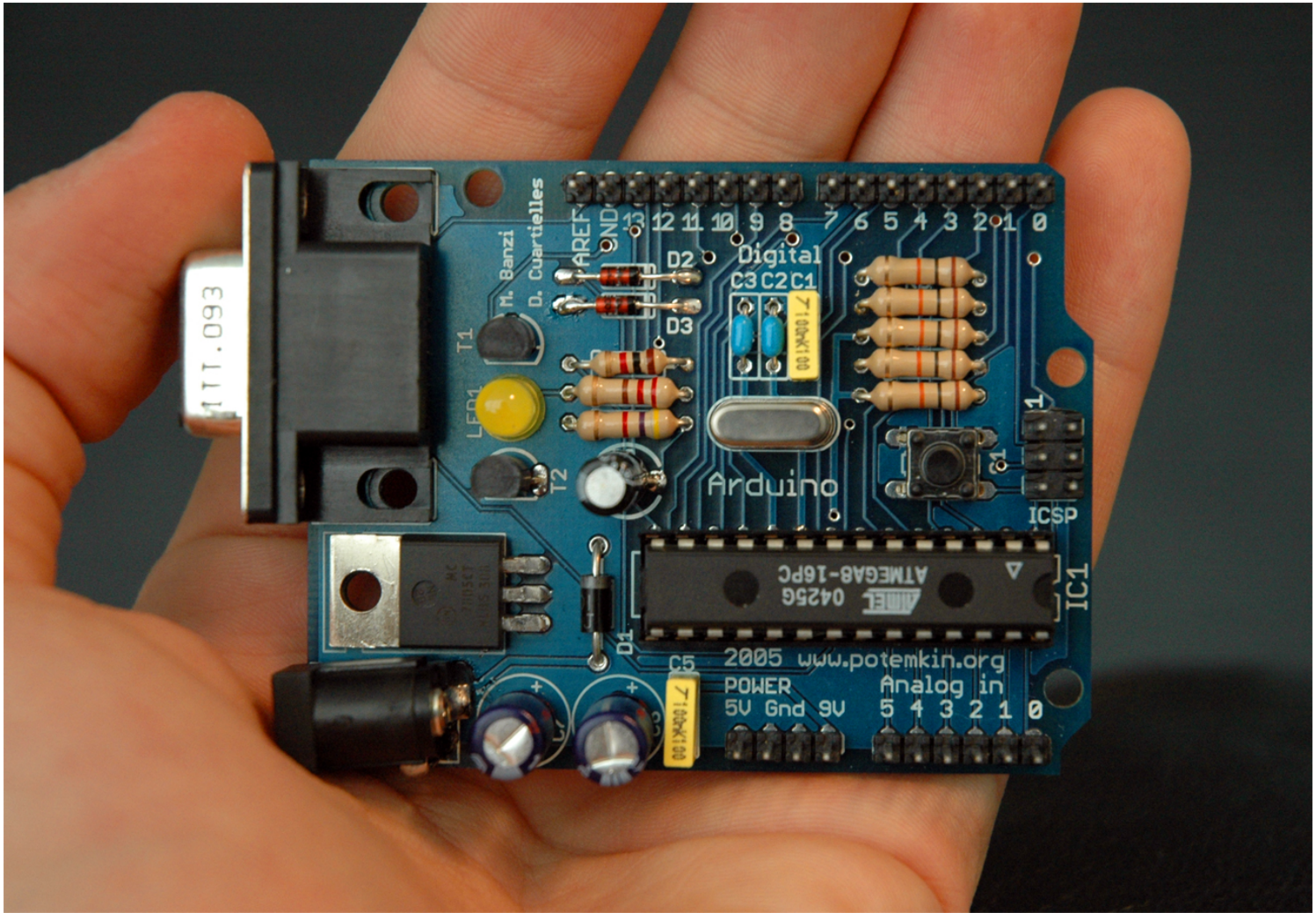
Objetivos do trabalho

Explorar limites e possibilidades de processamento de áudio em tempo real utilizando dispositivos acessíveis em termos de custo e tecnologia.

Estudos de caso:

- Microcontroladores: Arduino.
- Dispositivos móveis: Android OS.
- Processadores paralelos: GPU.

Arduino


M. Banzi
D. Cuartielles
AREF
GND
13 12 11 10 9 8
7 6 5 4 3 2 1 0
Digital
D2
D3
C3 C2 C1
T1
T2
LED1
Arduino
ICSP
ATMEGA8-16PC
0425G
IC1
D1
C5
2005 www.potemkin.org
POWER
5V Gnd 9V
Analog in
5 4 3 2 1 0

Arduino

ADC e DAC nativos no microcontrolador

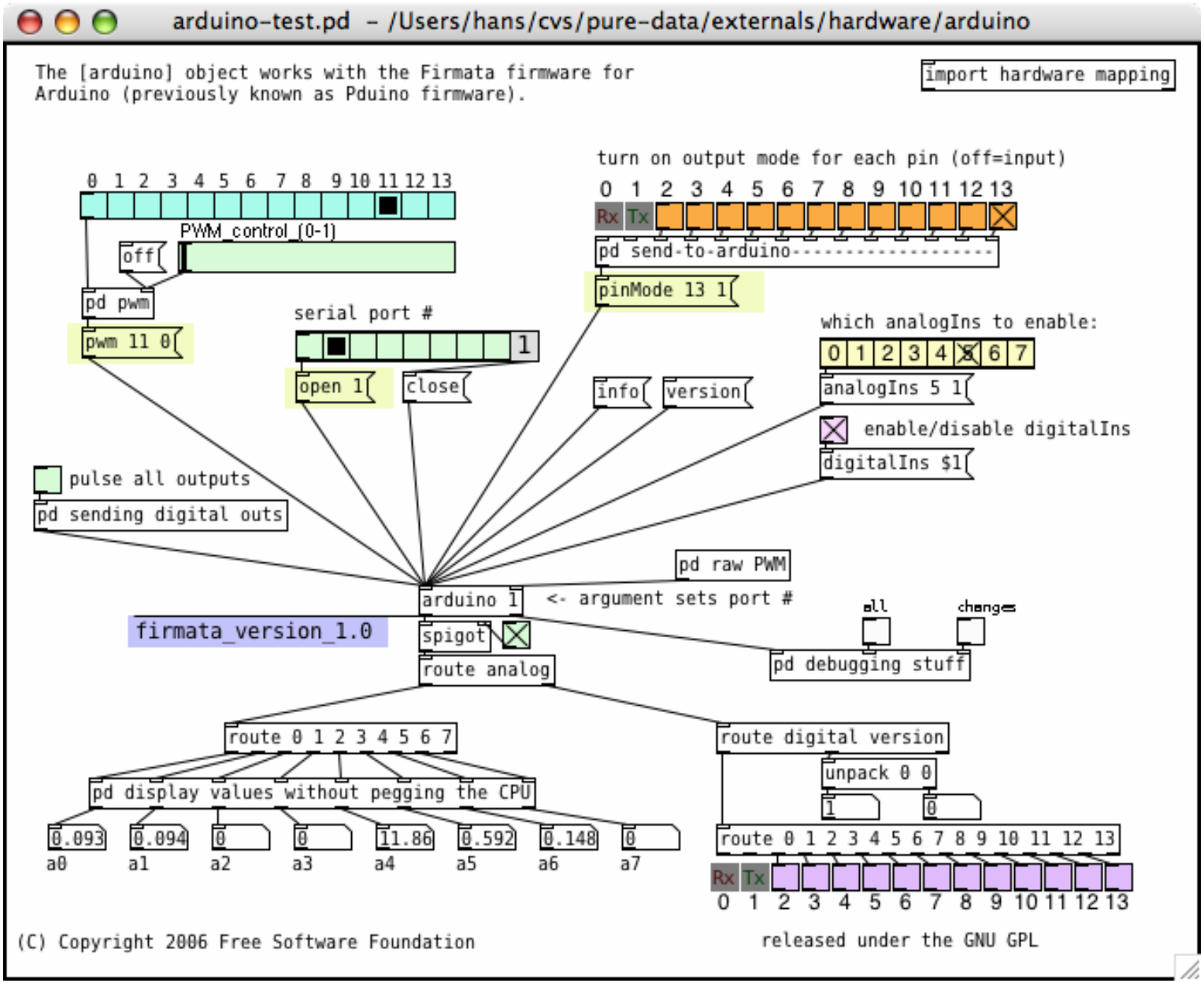
Amostragem de um sinal de entrada utilizando ADC do microcontrolador:

- Até 10 bits de resolução.
- 100 μs para obtenção de uma amostra.
- 10.000 Hz de taxa de amostragem.

Geração de sinais de áudio:

- PWM com resolução de 8 bits.
- Frequências até 500 Hz.

Arduino com Pure Data



Android

Figura: Android rodando em um tablet.

Android com Pure Data

A biblioteca **libpd** empacota as funções do Pd e:

- separa as funções de PDS da interface gráfica e de drivers;
- transforma o Pd em uma biblioteca de síntese e processamento de áudio; e
- permite a comunicação com código em outros ambientes.

Já existem versões para Android e iOS:

- http://gitorious.org/pdlib

GPU: Graphics Processing Unit

Figura: Placa com GPU.

Estrutura da apresentação

Processamento em tempo real utilizando dispositivos não convencionais

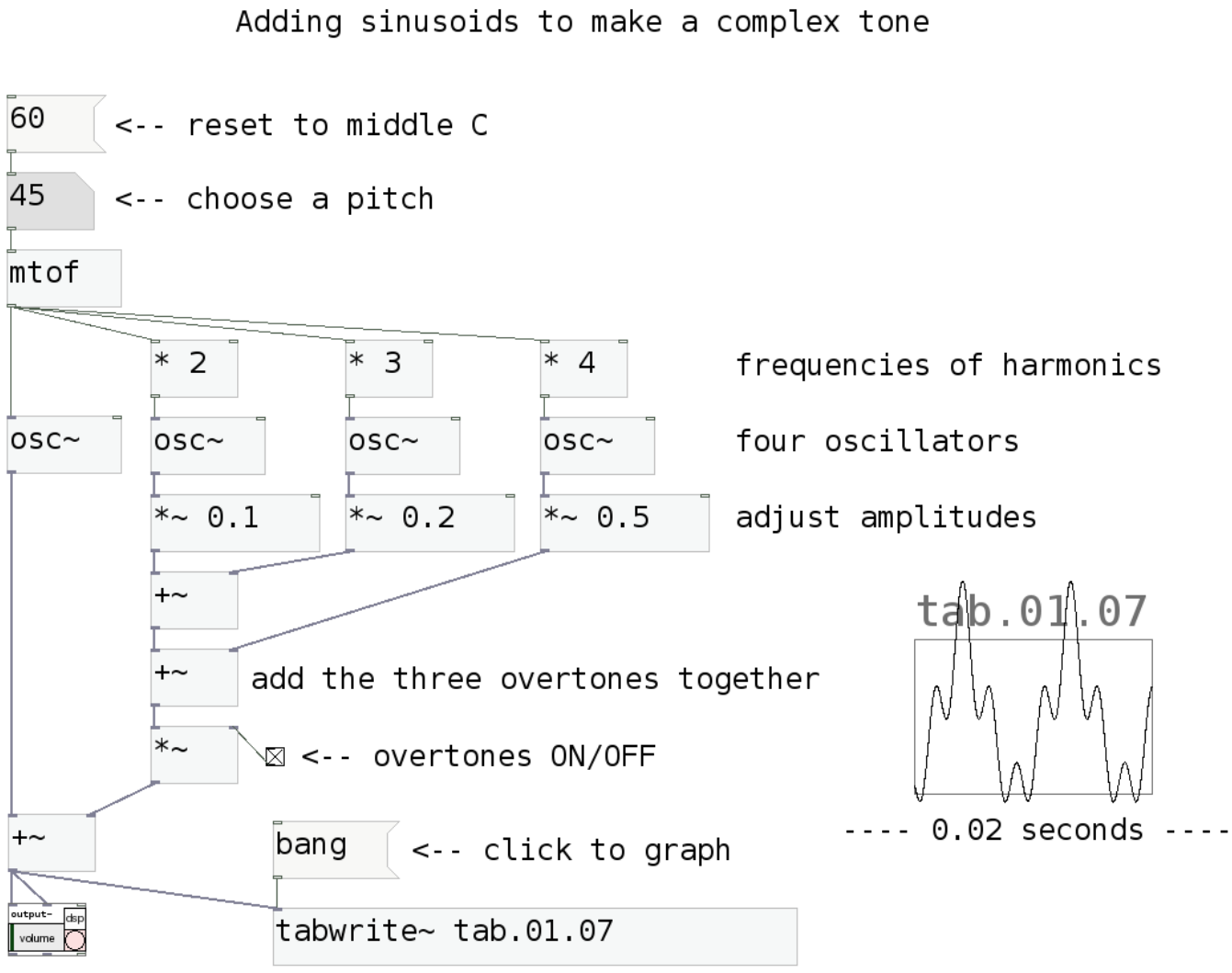
Pure Data (Pd)

GPU

Pd + GPU

Próximos passos

Pure Data (Pd)


Adding sinusoids to make a complex tone
60
<-- reset to middle C
45
<-- choose a pitch
mtof
* 2
* 3
* 4
frequencies of harmonics
osc~
osc~
osc~
osc~
four oscillators
*~ 0.1
*~ 0.2
*~ 0.5
adjust amplitudes
+~
+~
add the three overtones together
*~
<-- overtones ON/OFF
tab.01.07
---- 0.02 seconds ----
+~
bang
<-- click to graph
output~
dsp
volume
tabwrite~ tab.01.07

Pure Data (Pd)

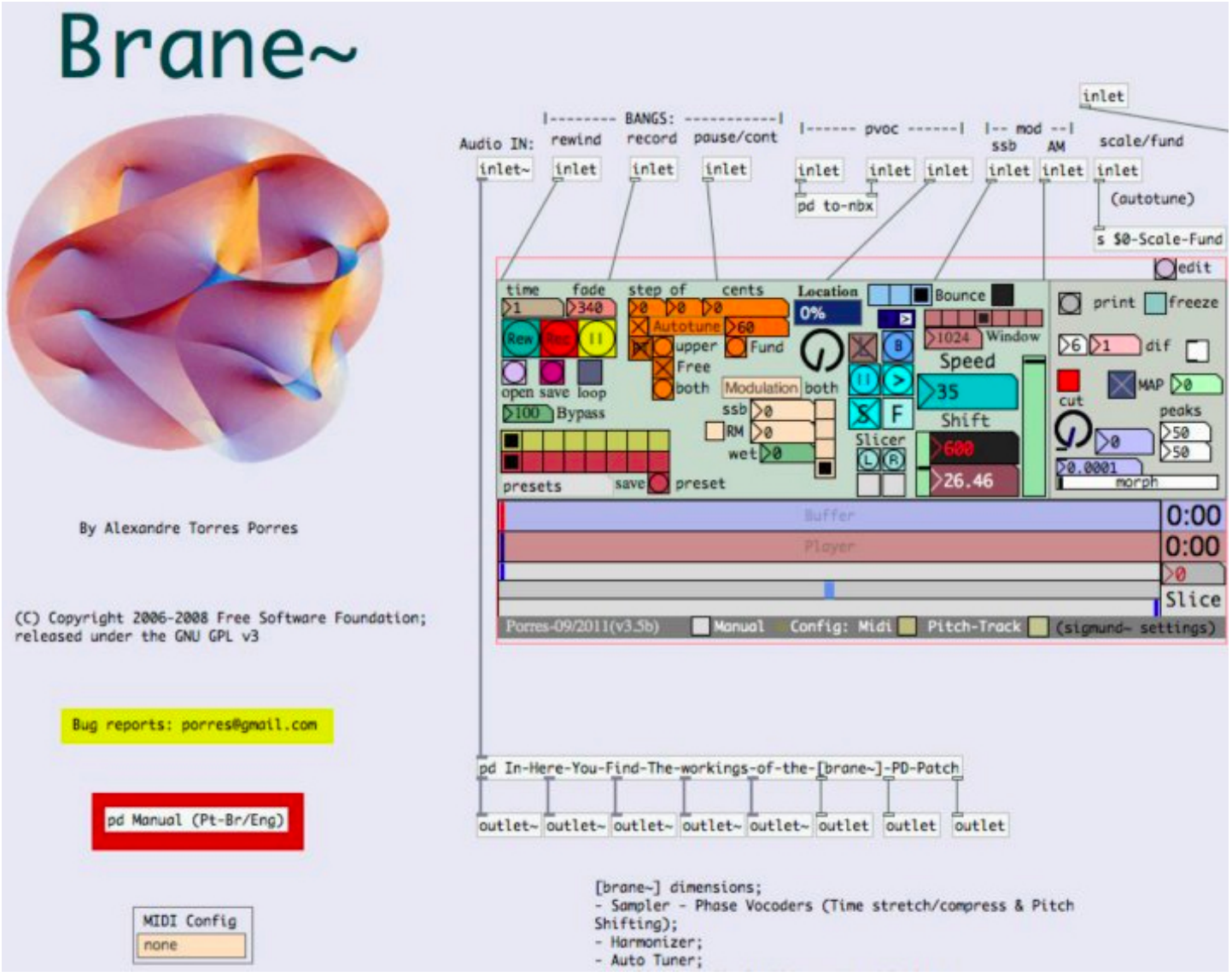


Figura: Phase Vocoder, sampler, autotuner e ressíntese com controle arbitrário de parciais em tempo real.

Pure Data (Pd)

Características do projeto:

- Extensível em C.
- Licença livre.
- Comunidade (suporte, código e manutenção).

Características da ferramenta:

- Processamento em tempo real.
- Interface com inúmeros dispositivos.
- Trabalha com áudio, imagem e vídeo.

Pure Data (Pd)

Características do processamento em tempo real no Pd:

- Os sinais fluem através de objetos e são modificados.
- A computação é realizada em blocos com fatores de sobreposição e reamostragem.
- Um "contexto PDS" é armazenado para cada tela.
- Algoritmo de ordenação das rotinas de PDS.
- Memória compartilhada para execução das funções de PDS.
- Alocação de sinais para os *inlets* e *outlets* de cada objeto, compartilhando uma mesma porção de memória sempre que possível.
- Transferência de memória ocorre somente quando existe alteração no tamanho do bloco ou na taxa de amostragem.

Pd e paralelismo

Motivações para o processamento paralelo de sinais no Pd:

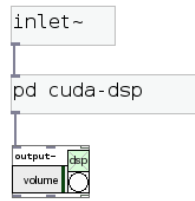
- Implementação do modelo de processamento de fluxos de dados.
- Alta taxa de fluxo de dados.
- Processamento de grande número de canais.
- Processamento de blocos de dados grandes.
- Funções PDS do Pd com paralelismo de dados diretamente mapeável para GPU.
- Aplicação da tecnologia de forma seletiva de forma a trazer o maior benefício para cada aplicação.

Pd e paralelismo

Possibilidades de extensão:

- *Patches*.
- *Externals*.
- Modificação do código fonte.

Pd e CUDA



Projeto em andamento: PdCUDA.

- Interface de Pd com CUDA.
- GPGPU, processamento de fluxos de dados e Pd.
- Prova de conceito e avaliação de desempenho.

Estrutura da apresentação

GPU: Graphics Processing Unit

Pipeline gráfica

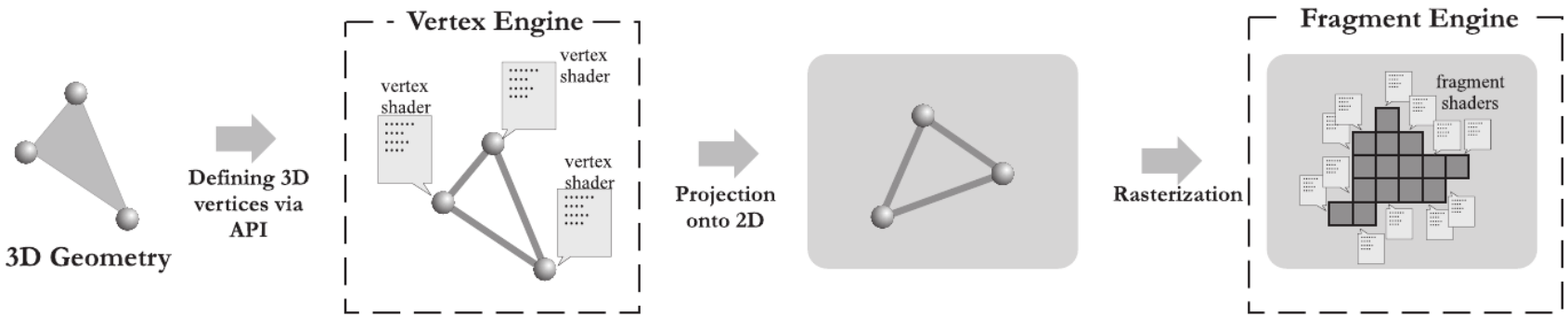


Figura: *Pipeline* de renderização 3D.

GPU: Graphics Processing Unit

Motivação: sistemas gráficos tradicionais

Características dos sistemas gráficos:

- Alto requerimento computacional.
- Alto grau de paralelismo.
- Alta taxa de fluxo de dados.

Questões críticas em sistemas gráficos:

- Computação versus comunicação.
- Computação versus controle.
- Paralelismo de dados e de tarefas.
- Balanço entre funções fixas e unidades programáveis.
- Performance versus flexibilidade.

GPU: Graphics Processing Unit

Comparação entre CPU e GPU

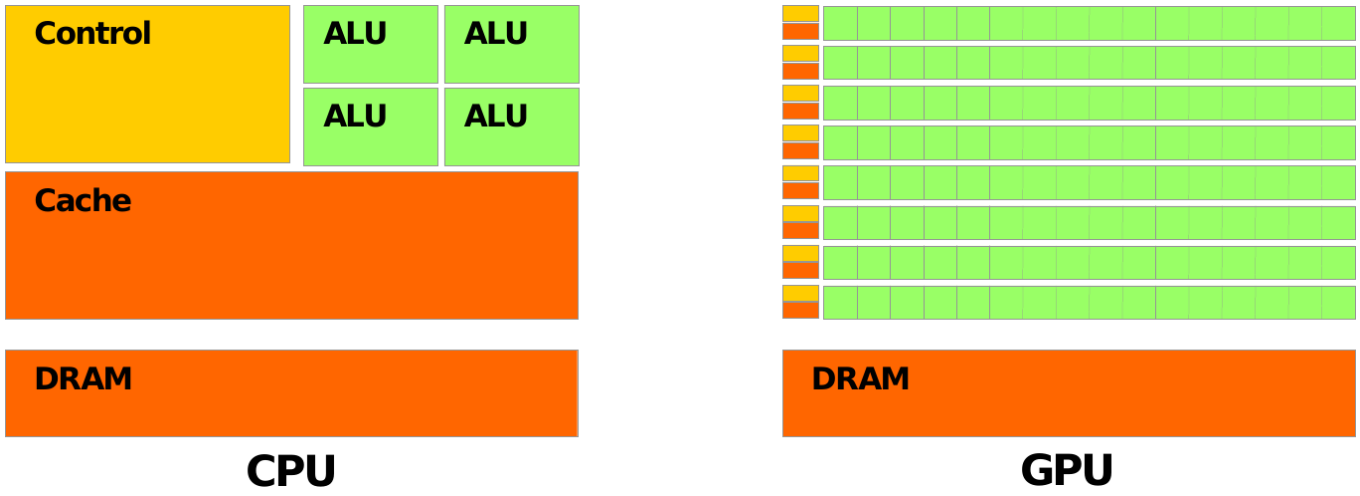
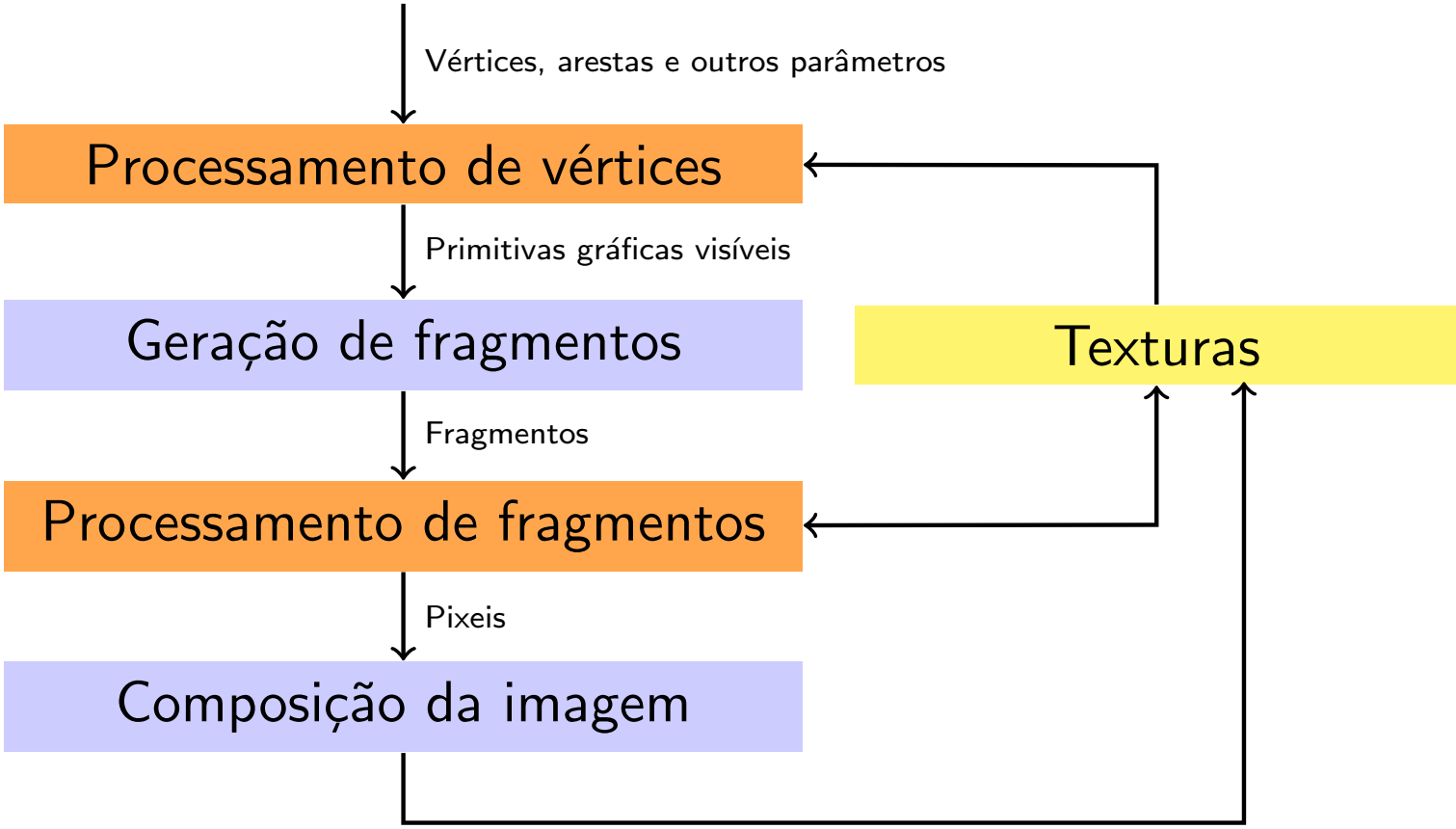


Figura: A GPU devota mais transistores para processamento do que para controle de fluxo e endereçamento.

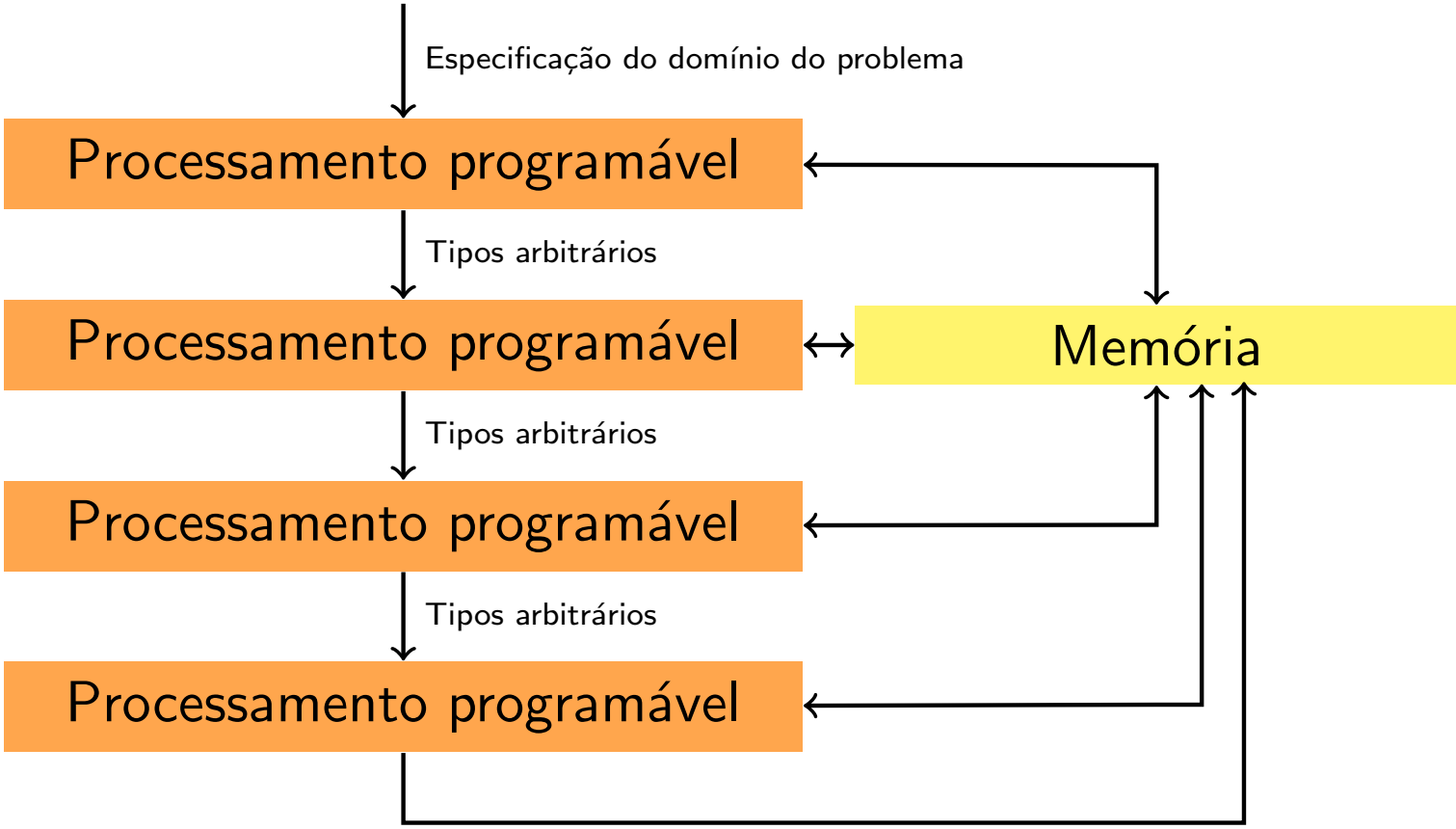
GPU: Graphics Processing Unit

Pipeline gráfica



GPU: Graphics Processing Unit

Processamento de propósito geral utilizando o modelo de fluxo de dados



Técnicas de programação para GPU (1/3)

GPU para gráficos

Geração de imagens:

- **Entrada:** vértices, arestas e texturas.
- **Processamento:** funções fixas e programáveis para processamento de vértices, primitivas e fragmentos.
- **Saída:** Fluxo de imagens para exibição na tela.

Técnicas de programação para GPU (2/3)

GPU para programação de propósito geral

Programação de propósito geral (modelo antigo):

- **Entrada:** vértices, arestas e texturas que representam tipos de dados em um domínio de computação de interesse.
- **Processamento:** funções fixas e programáveis para processamento de vértices, primitivas e fragmentos.
- **Saída:** áreas de memória com os resultados das operações.

Técnicas de programação para GPU (3/3)

GPU para programação de propósito geral

Programação de propósito geral (modelo recente):

- **Entrada:** especificação da computação através de modelo de fluxo de dados.
- **Processamento:** aplicação das funções nos fluxos de entrada utilizando paralelismo de tarefas e de dados.
- **Saída:** fluxo de dados resultante.

GPGPU

Problemas adaptados para solução com GPU

Exemplos de domínios mapeados para GPU:

- Processamento de sinais.
- Simulações biológicas.
- Simulações físicas.
- Métodos de álgebra linear.
- Métodos de equações diferenciais.
- Indexação e busca.

GPU: Processamento de sinais usando GPU.

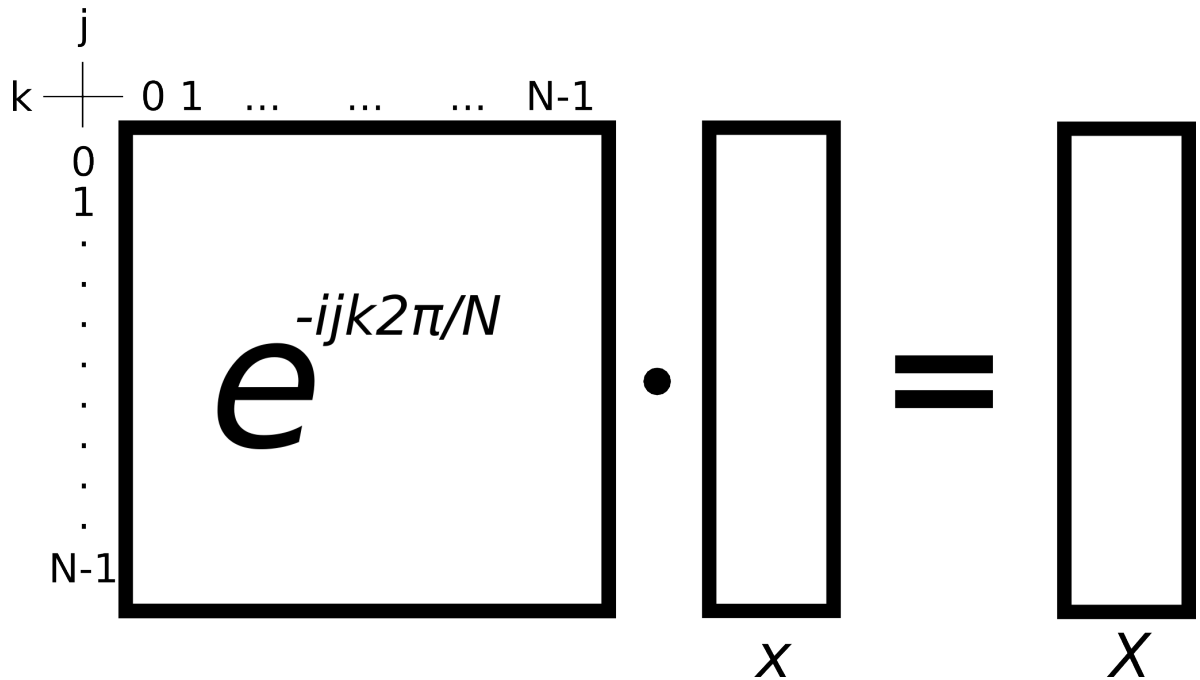
Trabalhos relacionados:

- FFT paralela.
- DCT paralela.
- DWT paralela.
- Áudio 3D.
- GPGPU e arcabouços para processamento de fluxos de dados.
- Integração com Pure Data.

Programação de propósito geral usando GPU

Exemplo: Transformada de Fourier

$$X_j = \sum_{k=0}^{N-1} x_k . e^{-ijk2\pi/N}$$



GPU: Graphics Processing Unit

Propostas de estudo

Processamentos computacionalmente pesados:

- Morphing em tempo real.
- Phase Vocoder com análise e ressíntese em tempo real.
- Auralização utilizando respostas impulsivas medidas ou simulação através de modelos geométricos.

Estrutura da apresentação

Pd + GPU

PdCUDA

Características da plataforma CUDA:

- Voltada para GPGPU.
- Placas de vídeo $\rightarrow$ coprocessadores paralelos.
- Aritmética de vetores e lógica organizada em *blocos* de *threads*.
- Hardware programável para operações de ponto flutuante.
- Foi escolhida para este projeto por ser mais adequado para GPGPU, menos geral (e portanto menos complexo) que OpenCL, mas facilmente adaptável.

Pd + GPU

PdCUDA

Objetivos do projeto PdCUDA:

- Permitir ajustes para melhor desempenho de aplicações específicas.
- Gerenciar sistematicamente o desempenho sempre que for possível.
- Minimizar a transferência de dados e, se possível, executá-la concorrentemente com outras *threads*.
- Produzir uma interface de usuário que não seja ambígua.
- Adaptar o agendador PDS do Pd para levar em conta diferentes tipos de GPU.
- Medir o desempenho das rotinas em GPU.
- Atingir eficiência no desempenho do PDS.
- Utilizar o melhor tipo de memória para cada função (exemplo: osciladores de leitura a tabela utilizando texturas).
- Prover um ambiente para o desenvolvimento de *patches* e *externals*.

Pd + GPU

PdCUDA

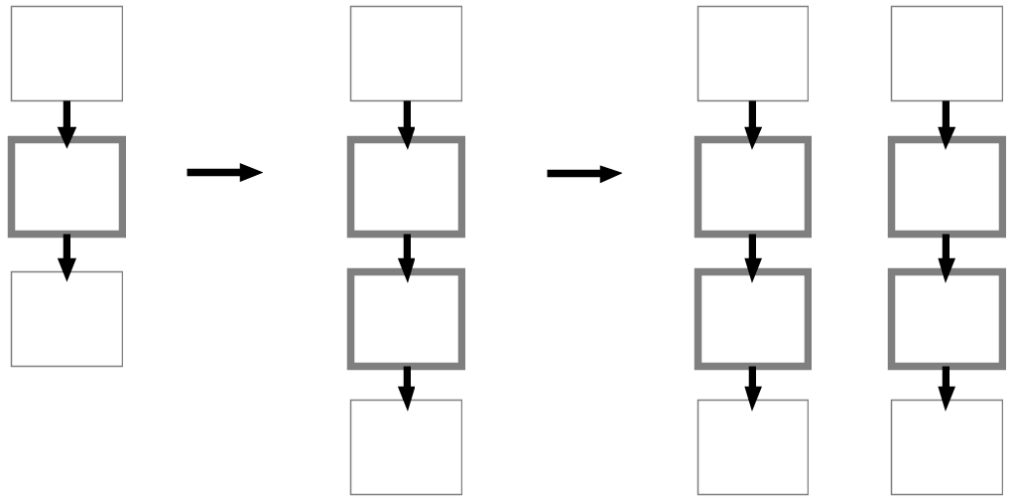


Figura: Casos de uso em ordem crescente de complexidade. Retângulos escuros representam funções executadas na GPU.

Pd + GPU

PdCUDA

Pontos importantes para implementação:

- Tradução de rotinas de PDS e estruturas de dados.
- Extensão do contexto PDS.
- Sobreposição de alocação de memória e computação.
- Modificações no agendamento das rotinas de PDS:
  - Busca em largura nas rotinas de alocação.
  - Busca em profundidade nas rotinas de computação, levando em conta a dependência da transferência de memória.
- Separação entre espaços de memória.
- Interface de usuário.

Pd + GPU

PdCUDA

Medição de desempenho:

- Taxa de processamento (*throughput*): GFLOPS.
- Uso de memória: MB.
- Taxa de transferência de dados: MB/s.
- Latência na transferência de memória: μs.
- Instruções por ciclo.
- Taxas de *cache-hits* e *cache-misses*.
- Instruções por byte.

Estrutura da apresentação

Pd + GPU

Próximos passos

- Rodar o código numa placa NVIDIA.
- Implementação de uma interface para visualização da análise de desempenho.

Obrigado pela atenção.

Dados de contato:

- Meu email: ajb@ime.usp.br
- Esta apresentação: http://www.ime.usp.br/~ajb/
- CM no IME: http://compmus.ime.usp.br/