Matlab的GPU加速计算简介

2013.12

一、GPU加速概念和原理

什么是Matlab?

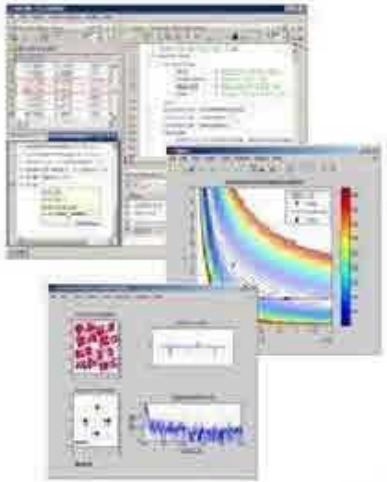
MathWorks MATLAB&SIMULINK

Core MathWorks Products

MATLAB

The leading environment for technical computing

- The *de facto* industry-standard, high-level programming language for algorithm development
- Numeric computation
- Data analysis and visualization
- Toolboxes for signal and image processing, statistics, optimization, symbolic math, and other areas
- Foundation of MathWorks products



- 工作标准的高级编程语言
- 用于算法开发
- 数值计算
- 数据分析和可视化
- 集成了信号处理、图像处理、统计、优化、符号数学等工具箱

最新版本2013b

CPU. 计算机中的中央单元，负责计算，控制以及监管计算机的其他部件。CPU 处理数据位于计算机内存当中的逻辑和浮点操作。

GPU.原本用于图形渲染的可编程芯片。对于需要并行处理大规模数据的算法而言，GPU 的高度并行架构使它们比通用的 CPU 更有效率。

核心. CPU 和 GPU 芯片中的一个独立的计算单元。CPU 和 GPU 核心并不是等同的。GPU 核心执行专门的操作，而 CPU 核心主要用于通用程序。

CUDA. NVIDIA 并行计算技术，由并行计算架构，开发工具，函数库，以及用于 CPU 编程的编程指令构成。

多核服务器以及多线程技术使科学家，工程师以及财务分析师能够加快处理多个学科内的计算密集型应用。现在，另一种硬件承诺提供更高的计算性能，那就是 GPU。

GPU 最初用于加速图形渲染，现在越来越多地应用于科学计算。和传统的 CPU 只包括少数的几个核不同，GPU 由整型和浮点处理器组成的大规模并行矩阵以及专用的高速内存构成。如图 1 所示，一个典型的 GPU 包含数百个小型处理器。

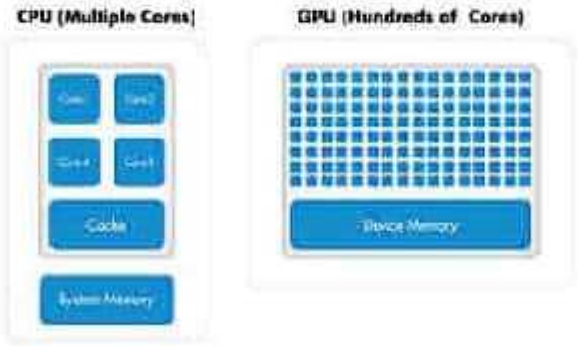
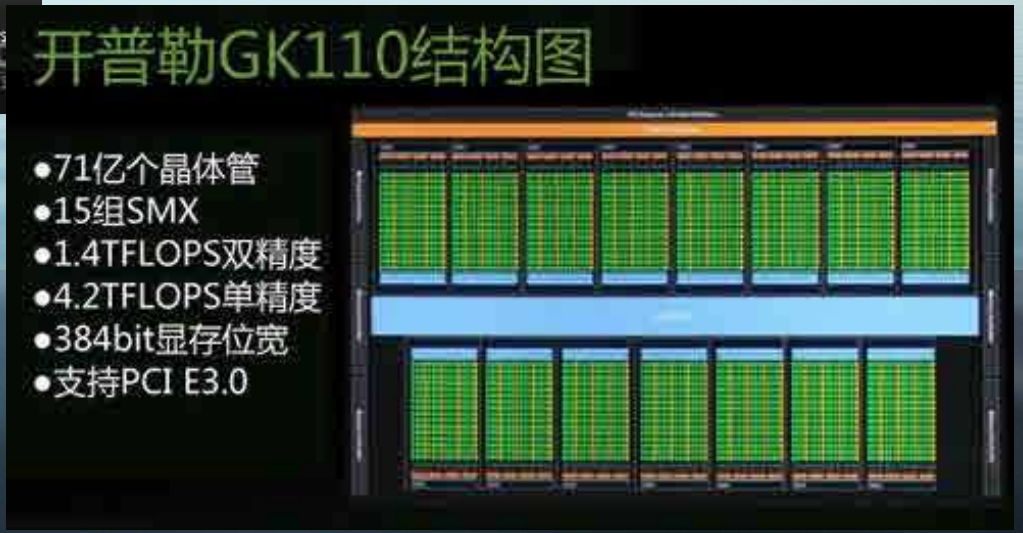


图 1. CPU 和 GPU 的核心数对比

核心数 448倍

浮点运算 14.2倍

晶体管 3.1倍



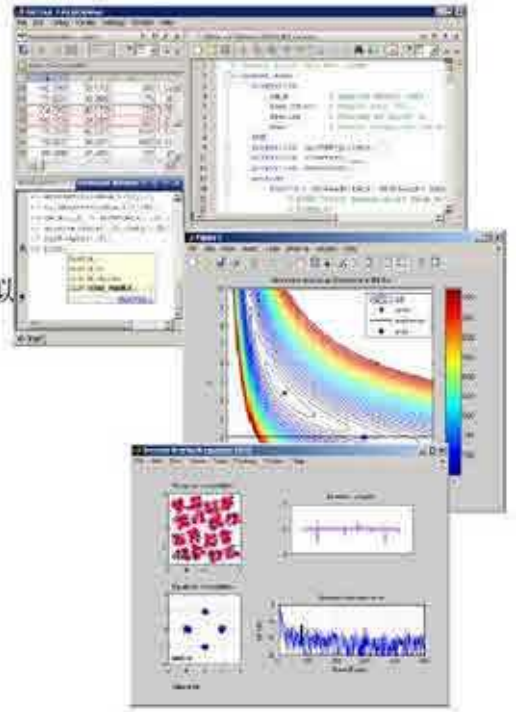
拥有超级计算机Tesla K20 一样的GPU

NVIDIA GeForce GTX TITAN

GTX TITAN 参数规格对比				
显卡	GTX TITAN	GTX690	GTX680	HD7970GE
代号	GK110	GK104	GK104	Tahiti XT
制作工艺	28nm	28nm	28nm	28nm
晶体管数量	71亿	2*35亿	35亿	43亿
流处理器	2688	2*1536	1536	2048
ROPs	48	2*32	32	32
纹理单元	224	2*128	128	128
核心频率	837-876MHz	915-1019MHz	1006-1058MHz	1000-1050MHz
显存频率	6008MHz	6008MHz	6008MHz	6000MHz
显存规格	6GB GDDR5	2*2GB GDDR5	2GB GDDR5	3GB GDDR5
显存位宽	384 bit	2*256bit	256 bit	384 bit
TDP	250W	300W	195W	240W
供电接口	8+6pin	8+8pin	6+6pin	8+6pin

MATLAB 并行计算工具箱

工业标准的算法开发和数据分析的高级语言



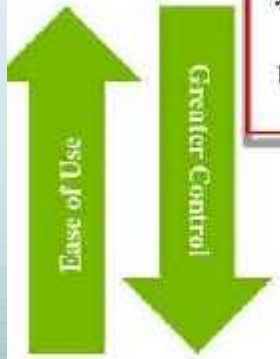
GPU 价值

- GPU可以加速现有的MATLAB的代码 对大数据集可以提高 10-40x 的性能
- datasets 对许多工作流有显著的加速，包括频谱分析，线性代数，随机模拟，
- 等等

亮点

- GPU 加速纯 MATLAB 操作
- 支持用户的 CUDA kernel 集成进MATLAB 支持MATLAB 编译器 (GPU 加
- 速不需要 MATLAB) 可以从GPU工作站(*Parallel Computing Toolbox*)
- 扩展到GPU集群 (*MATLAB Distributed Computing Server*

MATLABGPU加速的方式



在一台机器或集群的每一台机器上增加一块或多块**GPU**实现加速：

1) 使用 MATLAB 内置的 GPU array 接口函数

2) 在GPU array 中的元素执行客户函数

3) 利用现有的JCUDA代码和PT
kernel

+100 Functions support GPU Arrays

- **fft, fft2, ifft, ifft2**
- **Matrix multiplication (A*B)**
- **Matrix left division (A\b)**
- **LU factorization**
- **‘.’**
- **abs, acos, …, minus, …, plus, …, sin, …**

Matlab 超级计算工作站

工作站

AMAX PSC-2N， PSC-5N

Tesla卡

- Tesla C2070 (6 GB)
- Tesla C2075 (6GB)

新系统：

- CPU: 2 x 4core Xeon E5645
- Memory: 48GB
- Disk: 500 GB
- GPU: Quadro 600 + Tesla C2070/C2075

图像处理、雷达信号处理、通信信号处理、医学设备、生物研究、神经系统研究、气象建模、电力组网建模、病毒繁殖模型、基因序列研究

物理模拟、数学计算、金融计算、数据库、天然气和流体动力学、加密技术、**X**射线治疗、天文学、声音处理、计算机视觉、数据挖掘、数字电影和电视

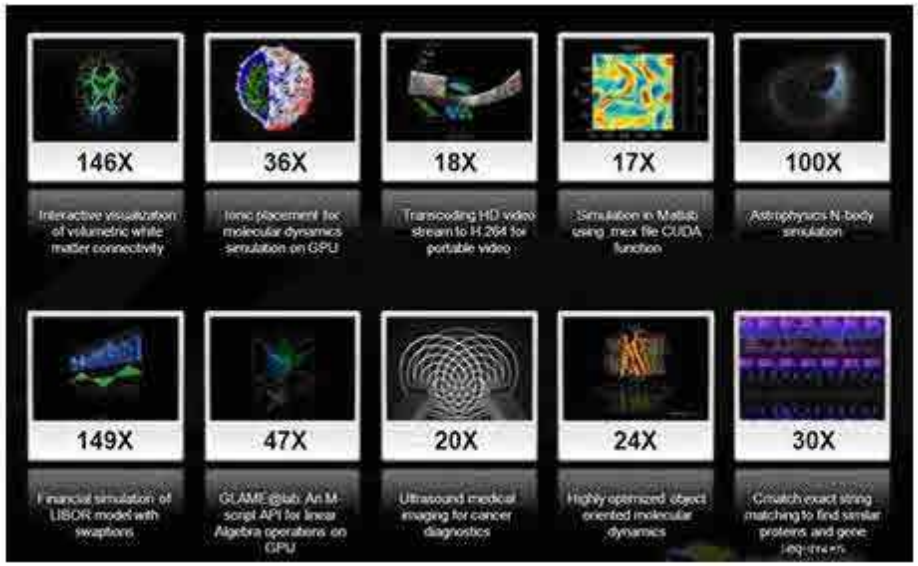


图6 不同项目使用CUDA后获得的加速比

geoinfo系统、军事应用、山区规划、分子动力学、磁共振成像、海洋学研究、粒子物理学、蛋白质凝集模拟、量子化学、射线追踪、可视化、油藏模拟、人工智能、卫星数据分析、地震勘探、外科、超声、视频会议等等多方面领域。

GPU Accelerated MATLAB Applications

GPU Enabled Functionality

Product Name	GPU Support	
Signal Processing Toolbox	1D Cross Correlation 2D Cross Correlation FFT Filtering Cross Covariance Circular Convolution	
Communications System Toolbox	LDPC Decoder Turbo Decoder Viterbi Decoder Convolutional Encoder PSK Demodulator PSK Modulator	Block Deinterleaver Block Interleaver Convolutional Deinterleaver Convolutional Interleaver AWGN Channel
Phased Array System Toolbox	Clutter simulation on GPU in an end to end airborne radar system	
Additional Products	All of products with GPU and parallel computing support	

二、性能比较

	Results for data-type 'double' (In GFLOPS)			Results for data-type 'single' (In GFLOPS)		
	MTimes	Backslash	FFT	MTimes	Backslash	FFT
GeForce GTX TITAN	**1403.08**	**483.61**	**154.77**	**3697.29**	**780.03**	**315.67**
Quadro K6000	1489.50	453.38	141.32	3998.82	737.72	295.48
Tesla K20c	1005.00	490.83	110.40	2690.21	772.21	257.51
Tesla C2075	327.83	242.26	69.13	684.97	425.15	144.56
GeForce GTX 680	139.20	97.53	58.82	1468.69	620.54	214.67
Host PC	**129.72**	**89.88**	**4.76**	**240.07**	**182.61**	**6.69**
Quadro 2000	38.60	33.01	14.18	232.90	122.57	46.32
GeForce GT 640	18.13	14.08	8.51	185.60	95.49	33.62
Quadro K600	13.24	10.69	6.17	135.40	0.01	26.57

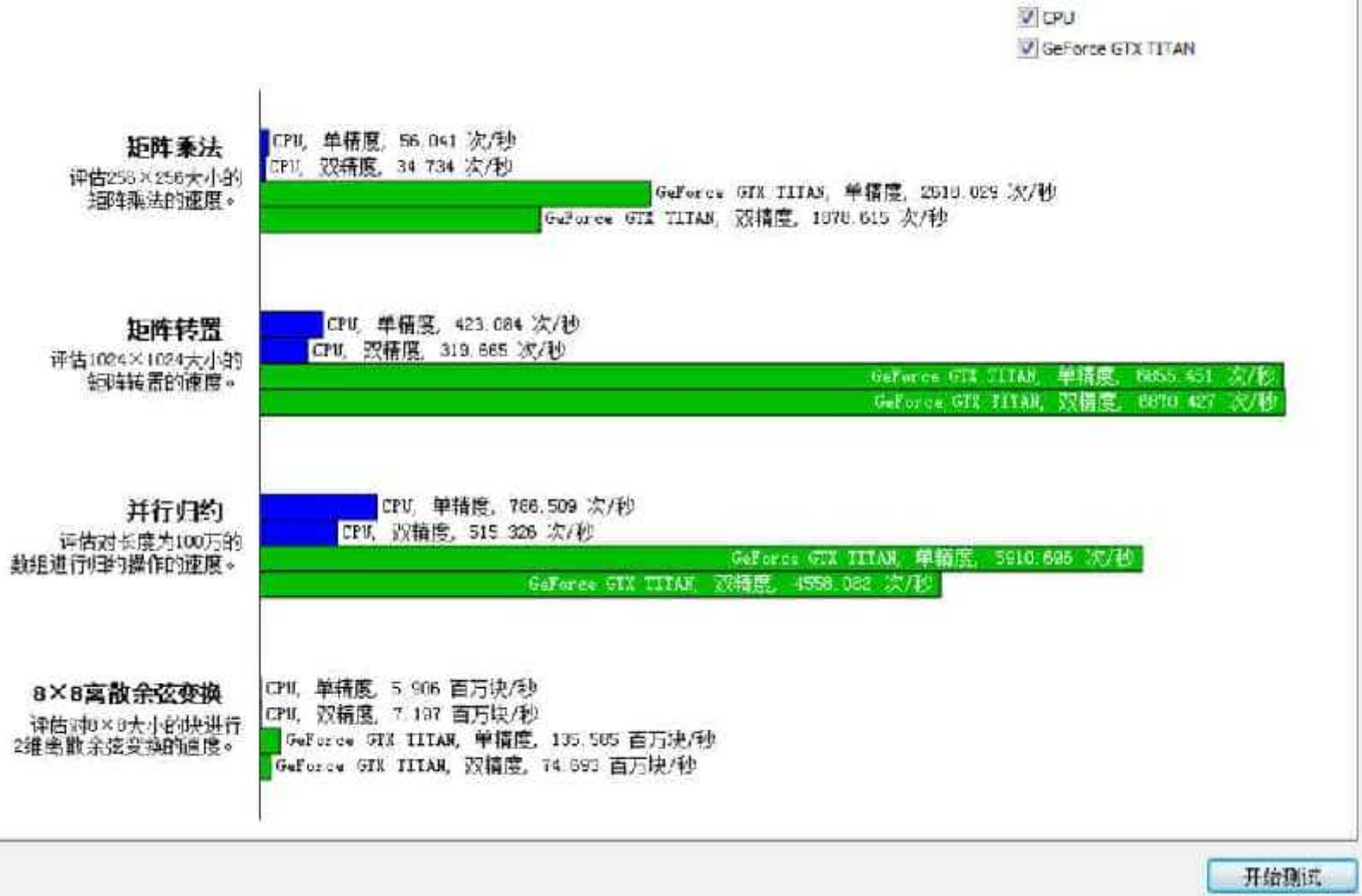
GTX TITAN VS i7-4770 （3.4GHz）

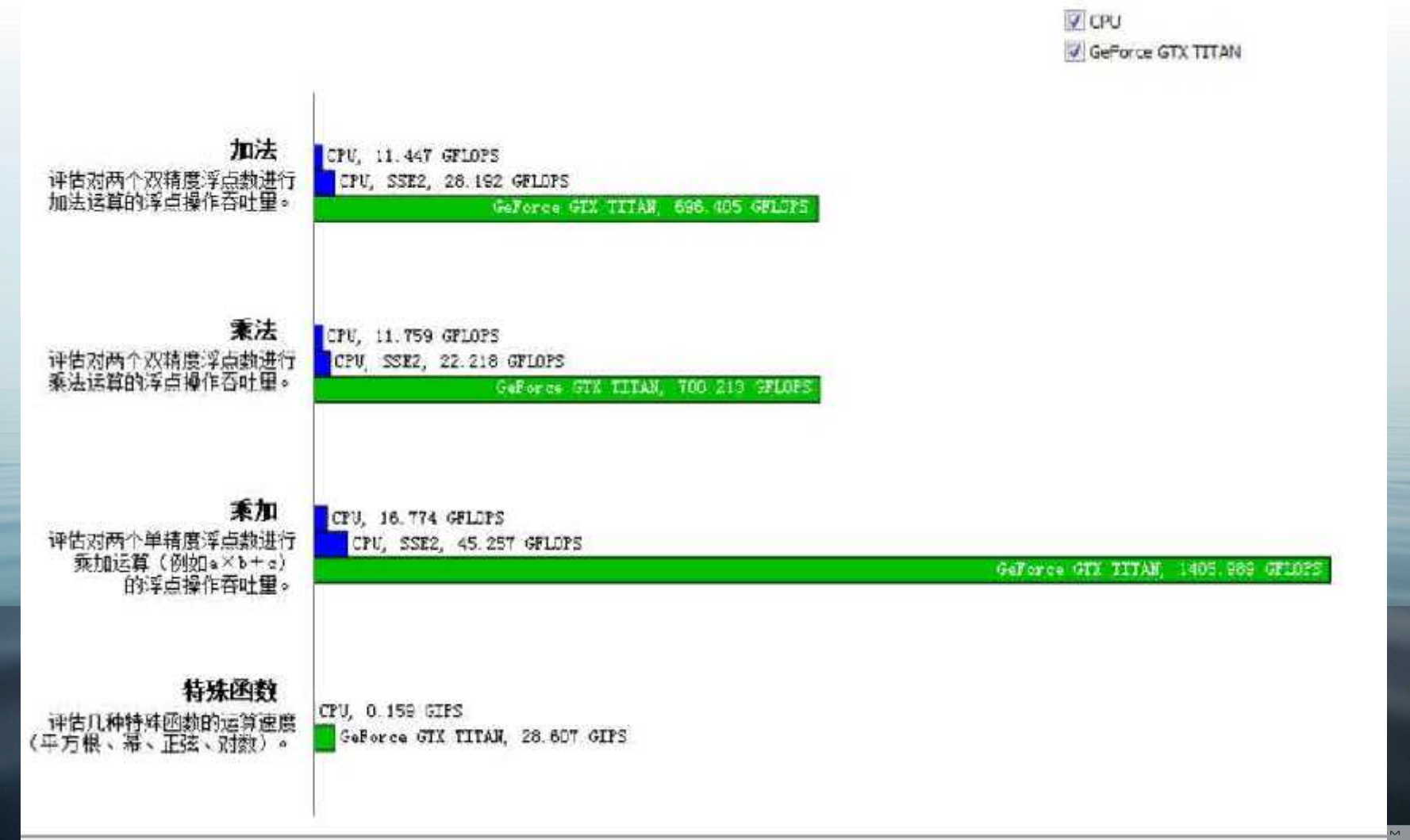
	双精度	单精度
矩阵乘法	11倍	15.4倍
矩阵除法	5.4倍	4.3倍
FFT	32.4倍	47倍

Benchmark Numbers | higher is better

http://www.expreview.com (last update: 2/22/2013)

Model	GeForce GTX TITAN	GeForce GTX 680	Different
Core/Shader/Memory Frequency	837/1502MHz	1006/1502	%
Memory Size	6144MB	2048MB	
Sandra 2012 GP Processing			
Native Float Shaders,Gpix/s	3.78	2.28	65.79%
Native Double Shaders,Gpix/s	1.85	0.197	839.09%
DirectCompute & OpenCL Benchmark			
OpenCL Score	11471.6	9832.8	16.67%
OpenCL GPC Bechmark			
Float Ops-MAD,GFLOPS	3570.432	2589.618	37.87%
Double Ops-MAD,GFLOPS	1390.026	134.898	930.43%
Cryption,MillonLopps/s	972.768	456.029	113.31%
Luxmark 2.0 | Room			
Score	?	291	
ComputeMark | 1024x600 | Fluid3D			
Score	7963	5837	36.42%
Total Average		**Compute Performance**	291.37%


CPU
GeForce GTX TITAN
矩阵乘法
评估256×256大小的矩阵乘法的速度。
CPU, 单精度, 56.041 次/秒
CPU, 双精度, 34.734 次/秒
GeForce GTX TITAN, 单精度, 2510.029 次/秒
GeForce GTX TITAN, 双精度, 1070.615 次/秒
矩阵转置
评估1024×1024大小的矩阵转置的速度。
CPU, 单精度, 423.084 次/秒
CPU, 双精度, 319.665 次/秒
GeForce GTX TITAN, 单精度, 6855.451 次/秒
GeForce GTX TITAN, 双精度, 6870.427 次/秒
并行归约
评估对长度为100万的数组进行归约操作的速度。
CPU, 单精度, 766.509 次/秒
CPU, 双精度, 515.326 次/秒
GeForce GTX TITAN, 单精度, 5910.695 次/秒
GeForce GTX TITAN, 双精度, 4558.082 次/秒
8×8离散余弦变换
评估对8×8大小的块进行2维离散余弦变换的速度。
CPU, 单精度, 5.906 百万块/秒
CPU, 双精度, 7.187 百万块/秒
GeForce GTX TITAN, 单精度, 135.505 百万块/秒
GeForce GTX TITAN, 双精度, 74.693 百万块/秒
开始测试


CPU
GeForce GTX TITAN
加法
评估对两个双精度浮点数进行加法运算的浮点操作吞吐量。
CPU, 11.447 GFLOPS
CPU, SSE2, 28.192 GFLOPS
GeForce GTX TITAN, 696.405 GFLOPS
乘法
评估对两个双精度浮点数进行乘法运算的浮点操作吞吐量。
CPU, 11.759 GFLOPS
CPU, SSE2, 22.218 GFLOPS
GeForce GTX TITAN, 700.213 GFLOPS
乘加
评估对两个单精度浮点数进行乘加运算（例如a×b+c）的浮点操作吞吐量。
CPU, 16.774 GFLOPS
CPU, SSE2, 45.257 GFLOPS
GeForce GTX TITAN, 1405.989 GFLOPS
特殊函数
评估几种特殊函数的运算速度（平方根、幂、正弦、对数）。
CPU, 0.159 GIPS
GeForce GTX TITAN, 28.607 GIPS


CPU
GeForce GTX TITAN
亮度直方图
评估亮度计算及直方图统计的速度。共评估3种方法：非原子操作、全局存储器原子操作和局部存储器原子操作。最快的会被记入最终成绩。
CPU, 1483.286 百万像素/秒
GeForce GTX TITAN, 非原子操作, 2636.963 百万像素/秒
GeForce GTX TITAN, 全局存储器原子操作, 0794.030 百万像素/秒
GeForce GTX TITAN, 局部存储器原子操作, 3224.781 百万像素/秒
2维卷积（锐化）
评估基于2维卷积的锐化操作的速度。共评估2种方法：全局存储器访问和纹理访问。最快的会被记入最终成绩。
CPU, 55.397 百万像素/秒
GeForce GTX TITAN, 全局存储器访问, 1606.585 百万像素/秒
GeForce GTX TITAN, 纹理访问, 2844.030 百万像素/秒
快速非局部均值法降噪
评估使用快速非局部均值法（块大小为7×7）进行降噪的速度。共评估2种方法：全局存储器访问和纹理访问。最快的会被记入最终成绩。
CPU, 7.173 百万像素/秒
GeForce GTX TITAN, 全局存储器访问, 291.395 百万像素/秒
GeForce GTX TITAN, 纹理访问, 0.000 百万像素/秒。不支持纹理访问。
缩放（双立方滤波）
评估使用双立方插值滤波进行缩放的速度。共评估2种方法：全局存储器访问和纹理访问。最快的会被记入最终成绩。
CPU, 45.027 百万像素/秒
GeForce GTX TITAN, 全局存储器访问, 2880.275 百万像素/秒
GeForce GTX TITAN, 纹理访问, 6451.137 百万像素/秒
开始测试

三、GPU加速在Matlab中的实现

实现方法

- 1、直接使用 GPUArray
-
- 使用简单、可使用Matlab原函数、提速一般

- 2、使用Matlab并行处理工具箱----PCT 或者 Jacket

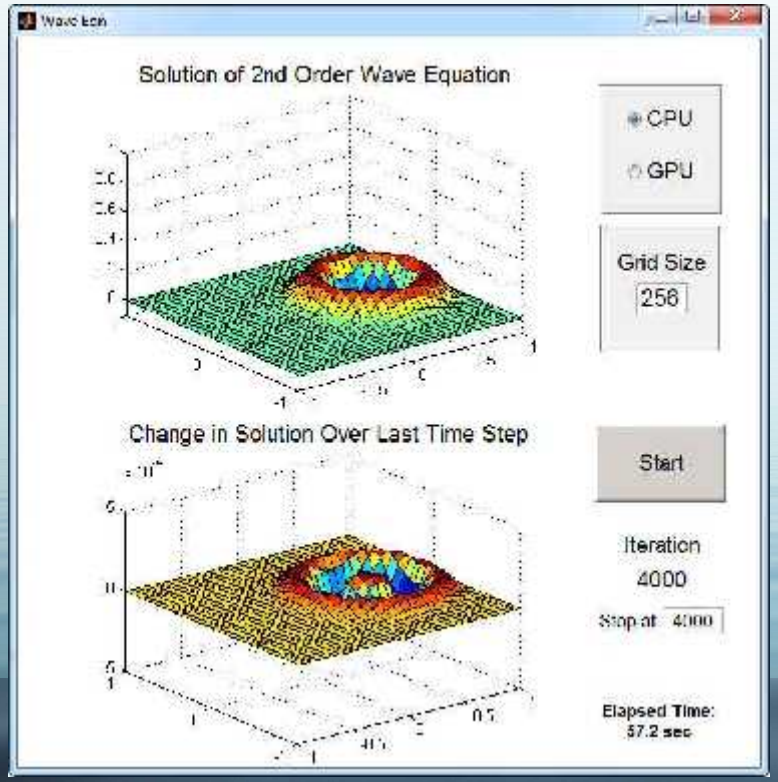
- 需要学习、使用自带函数、提速显著

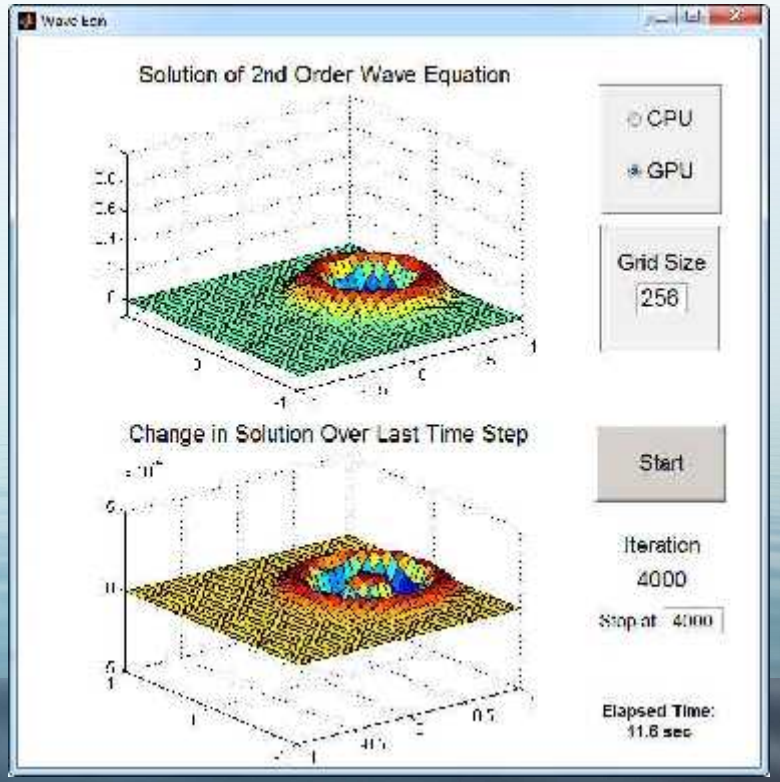
GPUArray

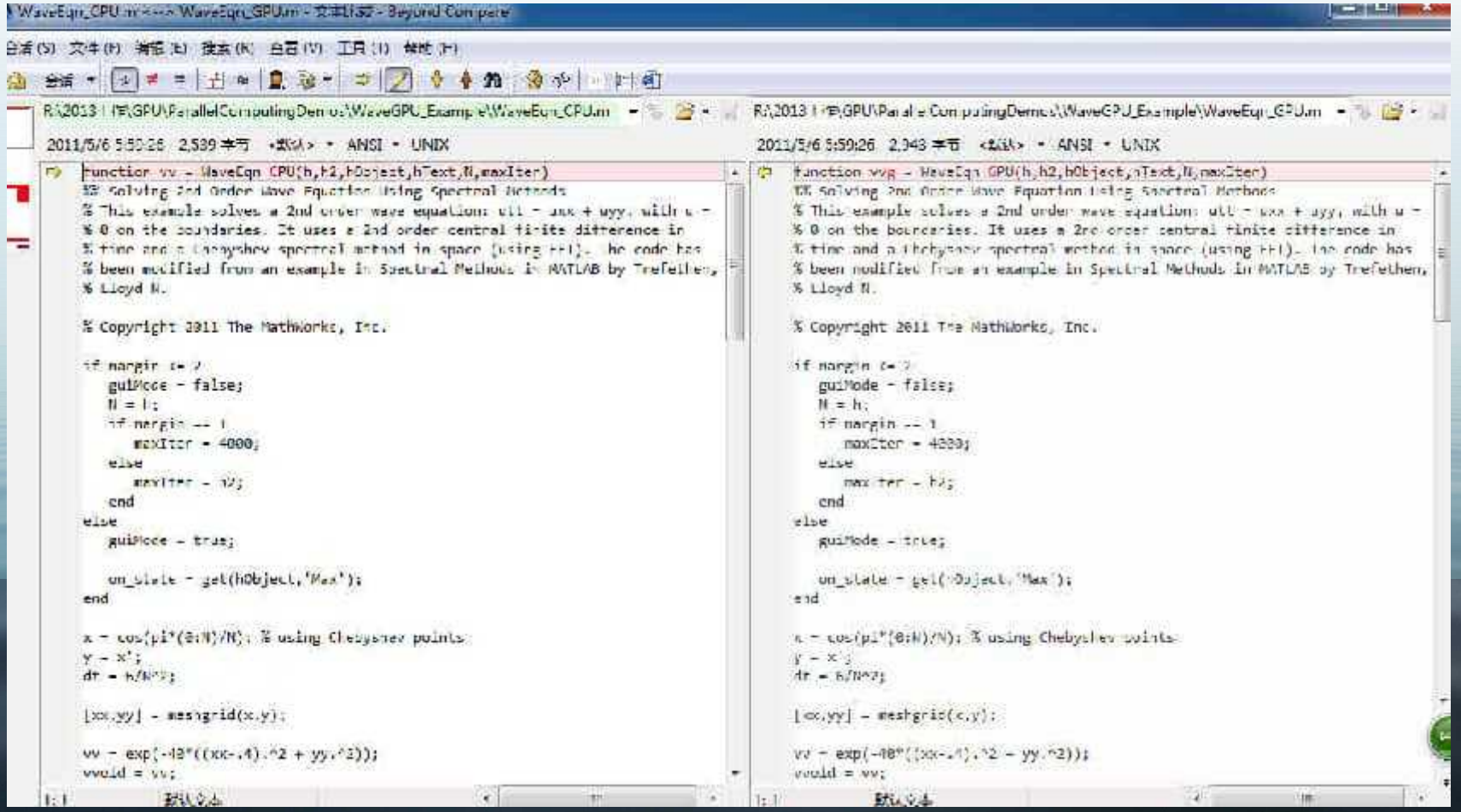
- GPUArray
- MATLAB中的GPUArray表示存储在GPU上的数据。使用gpuArray函数可以将数据从MATLAB工作空间传送到GPU。例如：
- A = data(10);
- G = gpuArray(A);

- 执行以上语句后、G 就是一个MATLAB GPUArray对象了。
- 当GPU运行完程序后、可以通过gather函数将数据从GPU取回到MATLAB工作空间。
- D = gather(G);

- GPUArray类也提供了以下静态方法、可用于直接在GPU上创建数组：
- parallel.gpu.GPUArray.ones
- parallel.gpu.GPUArray.eye
- parallel.gpu.GPUArray.zeros
- parallel.gpu.GPUArray.colon
- parallel.gpu.GPUArray.Inf
- parallel.gpu.GPUArray.true
- parallel.gpu.GPUArray.NaN
- parallel.gpu.GPUArray.false
- 以下几个函数可直接用于获取GPUArray对象的特性：
- classUnderlying、 isreal、 length、 ndims、 size





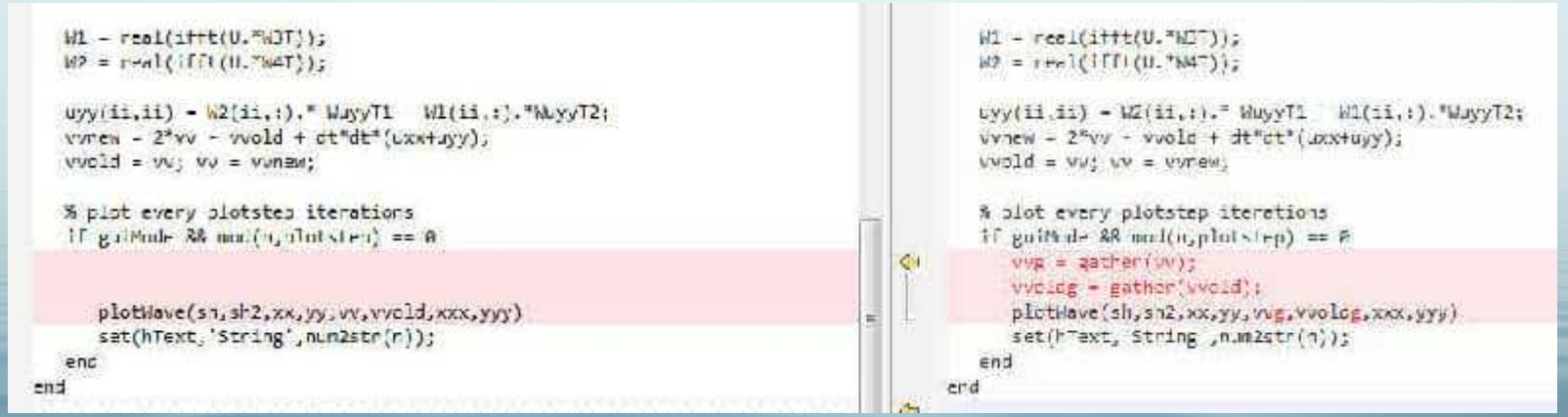

WaveEqn_CPU.m <---> WaveEqn_GPU.m - 文本比较 - Beyond Compare
会话(S) 文件(F) 编辑(E) 搜索(R) 查看(V) 工具(T) 帮助(H)
2011/5/6 5:59:26 2,539 字节 ANSI UNIX
2011/5/6 5:59:26 2,943 字节 ANSI UNIX
function vv = WaveEqn_CPU(h,h2,hObject,hText,N,maxIter)
% Solving 2nd Order Wave Equation Using Spectral Methods
% This example solves a 2nd order wave equation: utt = uxx + uyy, with u =
% 0 on the boundaries. It uses a 2nd order central finite difference in
% time and a Chebyshev spectral method in space (using fft). The code has
% been modified from an example in Spectral Methods in MATLAB by Trefethen,
% Lloyd N.
% Copyright 2011 The MathWorks, Inc.
if nargin < 2
guiMode = false;
N = 1;
if nargin == 1
maxIter = 4000;
else
maxIter = N;
end
else
guiMode = true;
un_state = get(hObject,'Max');
end
x = cos(pi*(0:N)/N); % using Chebyshev points
y = x';
dt = h/N^2;
[xx,yy] = meshgrid(x,y);
vv = exp(-40*((xx-.4).^2 + yy.^2));
vvold = vv;
function vvG = WaveEqn_GPU(h,h2,hObject,hText,N,maxIter)
% Solving 2nd Order Wave Equation Using Spectral Methods
% This example solves a 2nd order wave equation: utt = uxx + uyy, with u =
% 0 on the boundaries. It uses a 2nd order central finite difference in
% time and a Chebyshev spectral method in space (using fft). The code has
% been modified from an example in Spectral Methods in MATLAB by Trefethen,
% Lloyd N.
% Copyright 2011 The MathWorks, Inc.
if nargin < 2
guiMode = false;
N = h;
if nargin == 1
maxIter = 4000;
else
maxIter = h2;
end
else
guiMode = true;
un_state = get(hObject,'Max');
end
x = cos(pi*(0:N)/N); % using Chebyshev points
y = x';
dt = h/N^2;
[xx,yy] = meshgrid(x,y);
vv = exp(-40*((xx-.4).^2 + yy.^2));
vvold = vv;

2011/5/6 5:59:26 2,539 行 <既定> ANSI UNIX

```
M2T = repmat(index2,length(ii),1);
index3 = ii'[0:N-1 0 1 N:-1]';
M1 = repmat(index1,1,length(ii));
index4 = -[0:N-1 N:-1]'.^2;
M4T = repmat(index4,1,length(ii));

MuxxT1 = repmat((1./(1-x(ii).^2)),length(ii),1);
MuxxT2 = repmat(x(ii)./(1-x(ii).^2).^(3/2),length(ii),1);
MyyT1 = repmat(1./(1-y(ii).^2),1,length(ii));
MyyT2 = repmat(y(ii)./(1-y(ii).^2).^(3/2),1,length(ii));

uxx = zeros(N-1,N+1);
uyy = zeros(N-1,N+1);

% Start time-stepping
for n = 1:maxIter

  if guiMode && ~isequal(get(hObject, 'Value'), on_state)
    break;
  end
```

2011/5/6 5:59:26 2,943 行 <既定> ANSI UNIX

```
M2T = repmat(index2,length(ii),1);
index3 = ii'[0:N-1 0 1 N:-1]';
M1 = repmat(index1,1,length(ii));
index4 = -[0:N-1 N:-1]'.^2;
M4T = repmat(index4,1,length(ii));

MuxxT1 = repmat((1./(1-x(ii).^2)),length(ii),1);
MuxxT2 = repmat(x(ii)./(1-x(ii).^2).^(3/2),length(ii),1);
MyyT1 = repmat(1./(1-y(ii).^2),1,length(ii));
MyyT2 = repmat(y(ii)./(1-y(ii).^2).^(3/2),1,length(ii));

uxx = parallel.gpu.GPUArray.zeros(N-1,N+1);
uyy = parallel.gpu.GPUArray.zeros(N-1,N+1);

% Converting variables to gpuArrays to get variables in CPU memory
dt = gpuArray(dt);
vv = gpuArray(vv);
M1 = gpuArray(M1);
M2 = gpuArray(M2);
M2T = gpuArray(M2T);
M4T = gpuArray(M4T);
MuxxT1 = gpuArray(MuxxT1);
MuxxT2 = gpuArray(MuxxT2);
MyyT1 = gpuArray(MyyT1);
MyyT2 = gpuArray(MyyT2);

% Start time-stepping
for n = 1:maxIter

  if guiMode && ~isequal(get(hObject, 'Value'), on_state)
    break;
  end
```



```
    W1 = real(ifft(U.*WJT));
    W2 = real(ifft(U.*W4T));

    uyy(ii,ii) = W2(ii,:).* WuyyT1   W1(ii,:).*WuyyT2;
    vvnew = 2*vv - vvold + dt*dt*(uxx+uyy);
    vvold = vv; vv = vvnew;

    % plot every plotstep iterations
    if guiMode && mod(n,plotstep) == 0


        plotWave(sh,sh2,xx,yy,vv,vvold,xxx,yyy)
        set(hText,'String',num2str(n));
    end
end
```

```
    W1 = real(ifft(U.*WJT));
    W2 = real(ifft(U.*W4T));

    uyy(ii,ii) = W2(ii,:).* WuyyT1   W1(ii,:).*WuyyT2;
    vvnew = 2*vv - vvold + dt*dt*(uxx+uyy);
    vvold = vv; vv = vvnew;

    % plot every plotstep iterations
    if guiMode && mod(n,plotstep) == 0
        vvg = gather(vv);
        vvoldg = gather(vvold);
        plotWave(sh,sh2,xx,yy,vvg,vvoldg,xxx,yyy)
        set(hText,'String',num2str(n));
    end
end
```

JACKET函数

一、JACKET 函数

JACKET 与 MATLAB 整合

一旦 JACKET 安装完，它是透明的集成了 MATLAB 的用户界面，用户可以通过交互的 MATLAB 桌面和命令窗口以及写 M-函数使用 MATLAB 编辑器和调试器方式开始工作。所有 Jacket 数据以及任何其他 MATLAB 的矩阵在 MATLAB 的工作空间都是可见的。

Jacket GPU 的提供了 MATLAB 的 CPU 数据类型，如实数，复数的双精度，单精度数据，无符号 32 位整数,32 位整数，逻辑数据类型等等。任何变量在主机（CPU）的内存，可转换为存储在 JACKET 的 GPU 的数据类型。JACKET 的内存管理系统在 GPU 上的这些变量自动分配和管理。而在幕后。基于 GPU 的数据调用的任何函数将自动执行，实现动态编译，在 GPU 上无须任何额外的编程。

二、附录：JACKET 支持的函数

ABS 绝对价值和幅角
ACCUMARRAY-构建与积累阵列
ACOS- 反余弦，以弧度结果
ACOSD 反余弦，在度的结果
ACOSH-反余弦，在度的结果
ACOT-反余切，以弧度结果
ACOTD-反余切，以度结果
ACOTH-反双曲余切
ACSC-反余割，以弧度结果
ACSCD-反余割，以度结果
ACSCH-反双曲余割
AIRY- 艾里函数
ALL- **Determine whether all array elements are nonzero**
AND- 查找逻辑与输入的数组或标量
ANGLE 相角，相位角
ANY- 确定是否任何数组元素是非零
ARRAYFUN- 适用于函数的每个数组元素
ASEC- 逆割线，以弧度结果
ASECD- 逆割线，以度的结果
ASECH- 反双曲正割
ASIN- 逆正弦，以弧度结果
ASIND-逆正弦，以度为结果
J 虚数单位
KRON- 克罗内克张量积
LCM- 最小公倍数
LDIVIDE 左除
LDL- 埃尔米特矩阵分解
LE- 小于或等于
LEGENDRE- 伴随勒让德函数
LENGTH-向量的长
LINKDATA-自动更新时变量的变化图
LINSOLVE-求解线性方程组
LINSPACE-产生线性间隔向量数自然对数
LOG10- 普通（基 **10**）对数
LOG1P- 对于小 **x** 的值准确地计算 **log(1+x) LOG2- 2** 对数和相应的解剖浮点数到指数和尾数
LOGICAL-转换数字值逻辑
LOGM- 矩阵对数
LOGSPACE-对数间隔的向量生成
LSCOV- 最小二乘解的存在已知协方差 **LSQNONNEG**-非线性最小二乘约束问题
LT-小于
LU- LU 矩阵因子分解

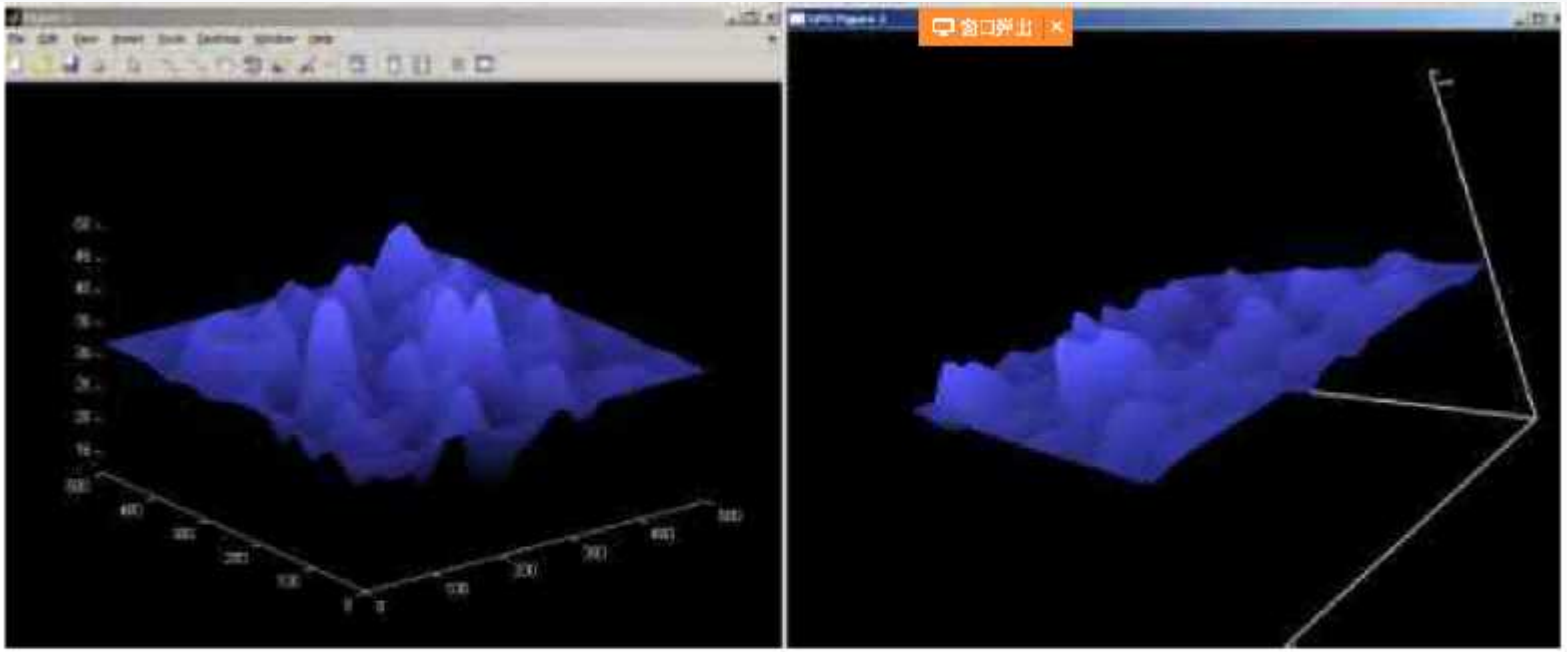
ASSIGNIN-变量分配值在指定工作区
ATAN- 逆切，以弧度结果
ATAN2-四象限反正切
ATAND- 逆切，以度为结果
ATANH- 反双曲正切
BALANCE- 对角线缩放以提高特征值的准确性
BESSELJ- 第一类贝塞尔函数
BESSELH- 贝塞尔函数或第三个类型（汉克尔函数）
BESSELI-修正贝塞尔函数的第一类型
BESSELK-修正贝塞尔函数第二类型
BESSELY- 第二类贝塞尔函数
BETA-贝塔函数
BETAINC- 不完全贝塔函数
BETALN- 对数函数的测试
BITAND-按位与
BITCMP- 按位补
BITGET 在指定的位置位
BITMAX 最高双精度浮点整数
BITOR- 按位或
BITSET-在指定位置设置位
BITSHIFT 一些地方指定的转移位
BITXOR- 按位异或
BLKDIAG- 构建块对角矩阵的输入参
MAX- 在数组中最大元素
MEAN- 平均或平均的数组值
MEDIAN- 中值数组
MESHGRID- **X** 和 **Y** 数组生成的三维图
MIN- 在数组中球最小元素
MINUS-一元减法或减
MKPP- 设为分段多项式
MLDIVIDE-反斜杠或矩阵左除
MOD- 取模
MODE- 最频繁的值在数组
MPOWER- 矩阵指数
MRDIVIDE- 斜线或矩阵权划分
MTIMES- 矩阵乘法
NAN- 非数
NCHOOSEK- 二项式系数或所有组合
NDGRID- 用于 **ND** 功能和插值生成数组 **NDIMS**- 数组维数
NE- 不等于
NEXTPOW2 返回第一个 **P** 使得 **2^ P> =abs（N）**
NNZ- 元素矩阵范数非零矢量和矩阵规范 **NORMEST**-**2** 范数估计
NOT 发现不符合逻辑数组或标量输入
NPPIABSDIFF_32F_C1R-**32** 位浮点，单通道绝对差

COMPLEX- 从实数构建复数数据。
COND- 关于条件数反转
CONDEIG- 关于条件数特征值
CONDEST 1 范条件数
CONJ-复共轭
CONTINUE- 控制传递给下一个迭代的 **for** 或 **while** 循环
CONV 卷积和多项式乘法
CONV2- 二维卷积
CONVN- **N** 维卷积
CORR2- 二维相关系数
CORRCOEF 相关系数
COS- 余弦弧度
COSD- 余弦度数
COSH-双曲余弦
COT 余切弧度
COTD- 余切度
COTH- 双曲余切
COV- 协方差矩阵
CPLXPAIR 成复杂的共轭复数对排序
CROSS- 矢量的叉积
CSC- 余割弧度
CSCD- 余割度
CSCH- 双曲余割
CTRANSPOSE- 矩阵转置
CUMPROD- 累积乘
蚀过滤
NPPIFILTER_8U_C1R- 综合运用线性二维卷积过滤器，缩放，**1** 通道 **8 bit/pixel** 图像
NPPIFILTERBOX_8U_C1R-找到一个在每个源 **1** 通道 **8-bit/pixel** 二维图像的像素面具地区当地平均像素值
NPPIHISTOGRAMEVEN_8U_C1R 计算直方图的 **8** 位单声道图像
NPPIMEAN_STDDEV_8U_C1R-计算 **8** 位单通道图像的像素值的平均值和标准差
NPPIMINMAX_8U_C1R-计算最低及 **8** 位单通道图像的像素值最高
NPPIMUL_32F_C1R- **32** 位无符号整数，单通道图像的乘法
NPPIMUL_8U_C1RSFS-**8** 位无符号整数，单通道图像的乘法
NPPINORMDIFF_INF_8U_C1R- 计算了两个 **8** 位单通道图像差异 **L_Inf** 规范
NPPINORMDIFF_L1_8U_C1R-计算了两个 **8** 位单通道图像差异 **L1** 范数
NPPINORMDIFF_L2_8U_C1R-计算了两个 **8** 位单通道图像差异 **L2** 范数
NPPISET_32F_C1R-**32** 位浮点，单通道图像数据集方法

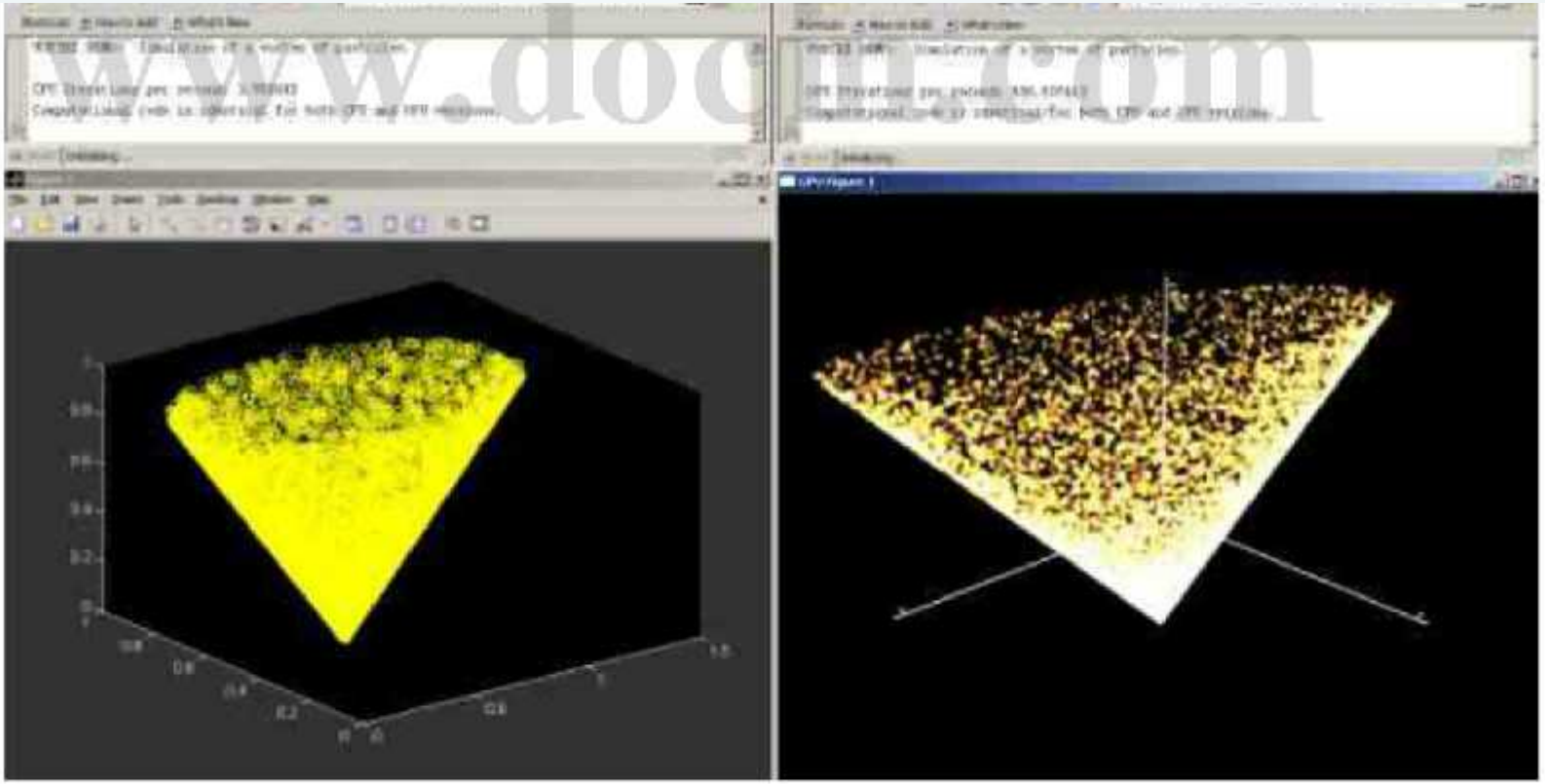
ELLIPKE- 第一类和第二类完全椭圆积分 **ELSE-** 执行语句，如果条件为假
ELSEIF-执行语句，如果是真正的附加条件 **END** 终止的代码块，或表示最后一个数组索引
EPS- 浮点相对精度
EQ- 测试是否相等
ERF- 误差函数
ERFC- 误差函数
ERFCX- 误差函数
ERFINV- 误差函数
ERFCINV- 误差函数
EVALIN- 在特定的工作空间执行 **MATLAB** 表达
EXIST- 检查变量，函数，目录，编程语言类 **EXP** 以 **e** 幂指数
EXPINT- 指数积分
EXPM- 矩阵指数
EXPM1-对于小的 **x** 值计算 **exp(x)-1**
EYE- 单位矩阵
FACTOR- 公因子
FACTORIAL- 阶乘函数
FALSE-逻辑 **0**（假）
FFT- 离散傅立叶变换
FFT2- 二维离散傅立叶变换
FFTN-N 维离散傅里叶变换
OR- 查找数组或标量逻辑或投入
ORDEIG- 按三角矩阵的特征值
ORDQZ-在 **QZ** 重新排序特征值分解
ORDSCHUR 在舒尔重新排序特征值分解 **ORTH-** 空间距离矩阵
PACK- 统一工作空间内存
PADECOEF- 延误时间
PCG-预条件共轭梯度法
PCHIP-分段三次 **Hermite** 插值多项式
PERMS- 所有可能的排列
PERMUTE- 重新排列尺寸阵列
PI- 圆的周长和直径之比
PINV- 穆尔-**Penrose** 逆矩阵
PLANEROT- 平面旋转
PLUS-单元加
POLY 多项式的根
POLYDER- 多项式导数
POLYEIG- 多项式特征值问题
POLYFIT- 多项式曲线拟合
POLYINT- 整合多项式解析
POLYVAL- 多项式求值
POLYVALM- 矩阵多项式求值
POW2- 以 **2** 为指数
POWER- 指数
PPVAL-分段多项式求值
PRIMES- 素数生成列表

GAMMAINC-伽玛函数
GCACHE-保存为特定的 GPU 编译 MATLAB 的脚本代码
GCD-最大公约数
GDOUBLE-转换为基于 GPU 的双精度
GE- 大于或等于
GEND- 结束同时运行多个 for 循环迭代在 GPU 上
GEVAL- 基于 GPU 计算和评估结果
GEYE- GPU 的单位矩阵
GFOR-同时运行多个 for 循环迭代在 GPU 上 GHELP- 在命令窗口帮助 JACKET 职能
GINFO- 运行时检测到的信息和图形卡 GINT32- 转换为符号的 32 位整数基于 GPU
GINT8- 转换为符号 8 位整数基于 GPU
GLAUNCH- 原型，执行和基准的 CUDA 内的 M-文件的内核
GLOAD 直接从磁盘加载到 GPU
GLOGICAL- 转换数值 GPU 为基础的逻辑 GMEX- 编译 GPU 的启用 mex 的 jacket SDK 的文件使用（在 Windows）
GONES- 生成一个基于 GPU 的阵列
GPROFILE- 测试 JACKET ,MATLAB
REALPOW- 阵列功率实时输出
REALSQRT-非负实矩阵的平方根
REM- 求余
REPMAT-阵列复制和旋转
RESHAPE- 重塑阵列
RESIDUE-部分分式扩张和转换系数多项式 RETURN- 返回到调用函数
ROOTS- 多项式的根
ROT90- 矩阵旋转 90 度
ROUND- 最接近的整数
RREF- 减少排梯队形式
RSF2CSF-真正的 Schur 形式转换到复杂的舒尔形式
SCHUR- 舒尔分解
SEC- 弧度割线
SECD- 度割线
SECH- 双曲正割
SETDIFF-找到两个向量的差集
SETXOR- 查找设置两个向量异或
SHIFTDIM 移尺寸
SIGN-正负号函数
SIN- 正弦弧度
SIND- 正弦度
SINGLE-转换为单精度
SINH- 双曲正弦论据弧度

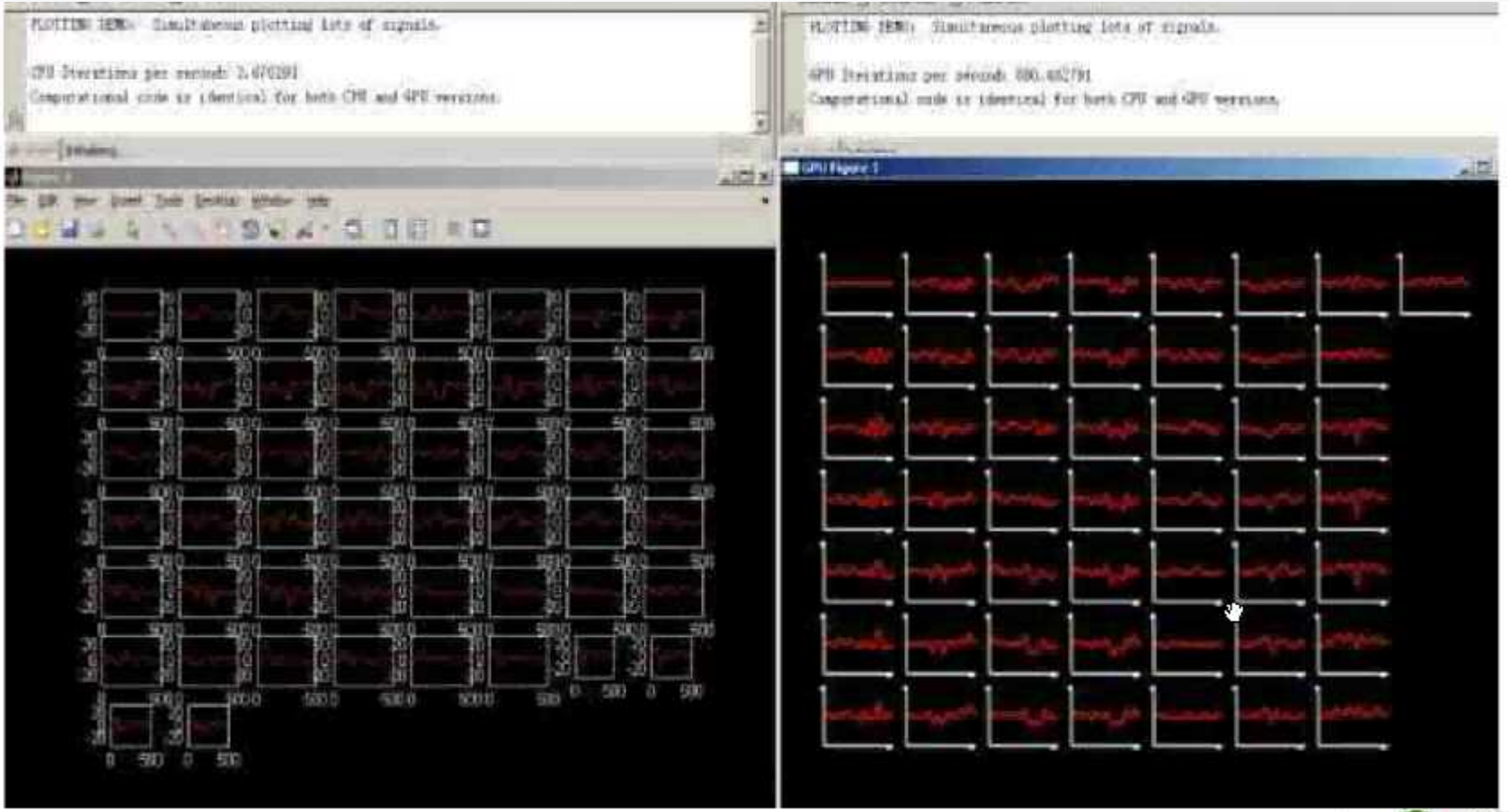
IFFT 逆离散傅里叶变换
IFFT2-二维离散傅里叶逆变换
IFFTN-逆离散傅里叶变换
IFFTSHIFT- 二维逆 **FFT** 的转变
ILU 稀疏不完全 **LU** 分解
IMAG- 复数的虚部
IND2SUB-从线性指数标-干扰素无穷
INT8- 转换为符号 **8** 位整数。
INT16 转换为符号的 **16** 位整数。
INT32-转换为符号 **32** 位整数。
INT64- 转换为符号 **64** 位整数。
INTERP1-**1**-**D** 数据插值（查表）
INTERP1Q 快速一维线性插值
INTERP2-**2**-**D** 数据插值（查表）
INTERP3-**3**-**D** 数据插值（查表）
INTERPFT- 一维插值采用 **FFT** 方法
INTERPN 数据插值
INTERSECT-查找设置两个向量交集
INTMAX- 指定的最大值整数类型
INTMIN- 最小的整数类型的值指定
INV 矩阵求逆
IPERMUTE- 逆置换维度
ISA-确定是否输入是给定类的对象
ISEMPTY 确定是否为空数组
ISEQUAL 判断是否相等
ISEQUALWITHEQUALNANS-测试阵
TYPECAST 数据类型转换不改变基础数据
UINT8-转换为无符号 **8** 位整数
UINT16-转换为无符号 **16** 位整数
UINT32 转换为无符号 **32** 位整数
UINT64- 转换为无符号 **64** 位整数
UNION-查找设置两个向量连接
UNIQUE-寻找独特的矢量元素
UNMKPP 分段多项式
UNWRAP- 正确的相位角，以产生平滑相图 **VAR**- 方差
VERTCAT- 连接垂直阵列
WHICH 定位功能和文件
WHILE-重复执行语句，而条件为真
WORKSPACE- 打开工作区浏览器来管理工作区
XOR 逻辑异或
ZEROS-创建零数组
ISSPARSE-确定输入资料是否稀疏
ISVARNAME-确定是否输入是有效的变量名
ISVECTOR- 确定是否输入向量



测试结果：在实时雨滴仿真实验中，利用 CPU 进行模拟，图像效果不真实并且不流畅，FPS（每秒渲染的帧数）平均仅为 9.200000；而在 GPU 上进行模拟，得到了非常有真实感的图像，画面流畅，并且 FPS 平均达到了 324.653417。此 36 倍的差异，无疑显示了使用 GPU 为 MATLAB 带来的超强的运算能力和图形显示能力。



测试结果：在对涡流模拟的测试中，利用 CPU 进行模拟，FPS 均值为 3.558445；而在 GPU 上进行模拟，FPS 均值达到了 900.607413，GPU 相对于 CPU 的加速比为 300。



加速了 192 倍

四、GPU加速的意义

- 1、加速、加速!
- 一个星期的运算可在一天内完成;
- 一整天（10小时）的运算可在一小时内完成。

- 2、与其他软件进行联合仿真、提高效率。如VC++、ADS、Cadence、Synopsys等。

- 3、实现实时的半物理仿真、代替真实硬件或仪器仪表。

谢 谢!

Make Presentation much more fun