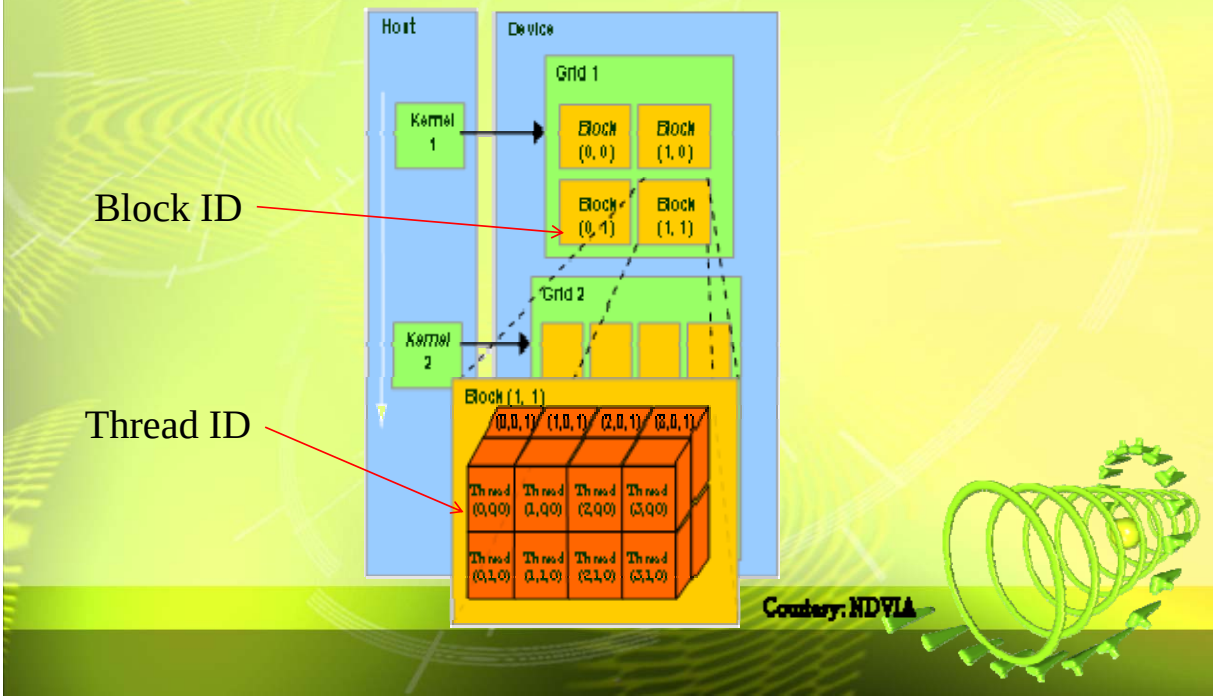
CUDA

Session 3

CUDA Threading Model

Hadi Salimi and Nima Ghaemian
Distributed Systems Lab.
School of Computer Engineering,
Iran University of Science and Technology,
hsalimi@iust.ac.ir and nima@comp.iust.ac.ir

Block IDs and Thread IDs



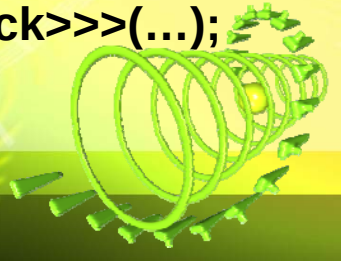
Block IDs and Thread IDs

- Each block in the array is labeled with (blockId.x, blockId.y)
- Each thread uses IDs to decide what data to work on:
  - Block ID: 1D or 2D
  - Thread ID: 1D, 2D, or 3D
- Simplifies memory addressing when processing multidimensional data:
  - Image processing
  - Solving PDEs on volumes
  - …

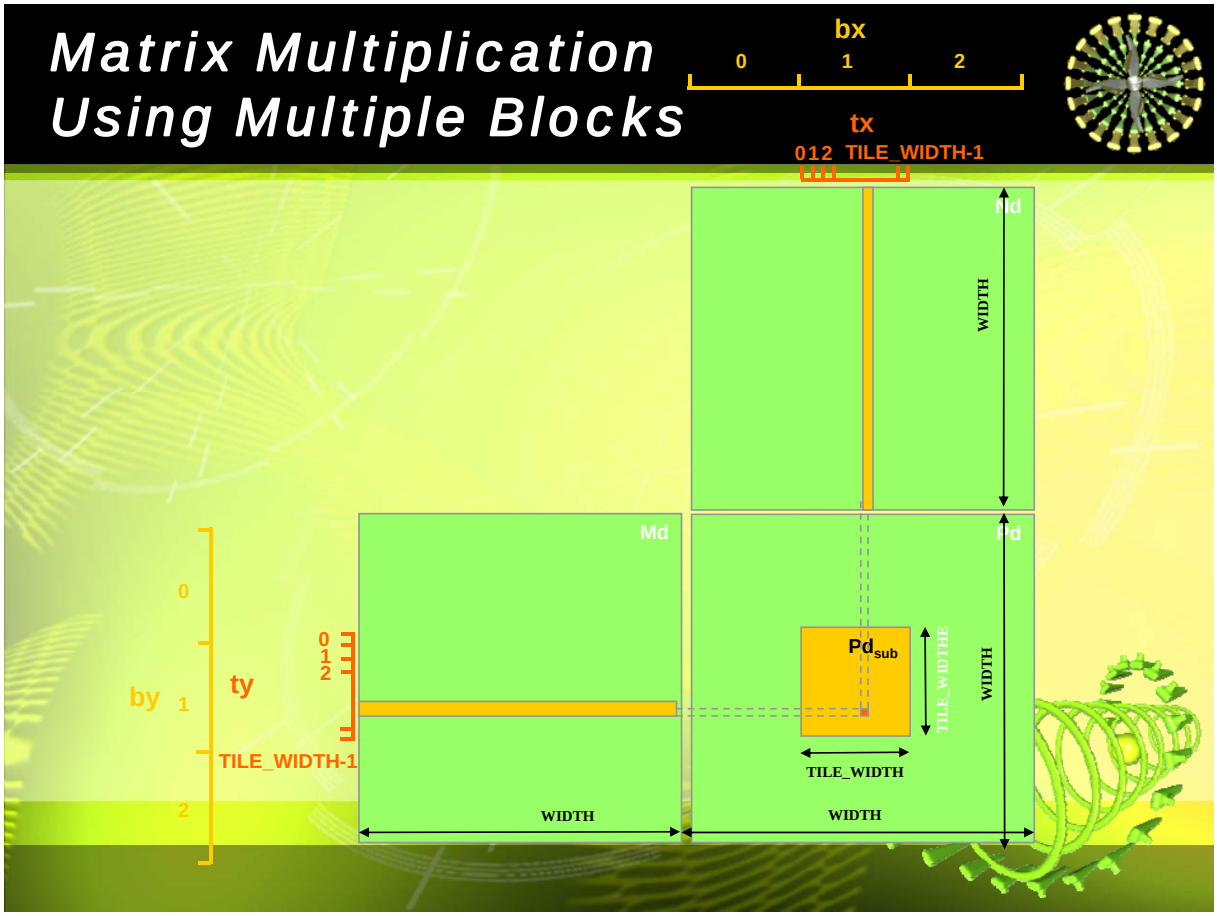
Example

- What is the form of the kernel launch of the thread organization that shown in the previous figure?
- The answer is:

```
dim3 dimBlock(4, 2, 2);
dim3 dimGrid(2, 2, 1); //for clarity!
KernelFunction<<<dimGrid, dimBlock>>>(…);
```



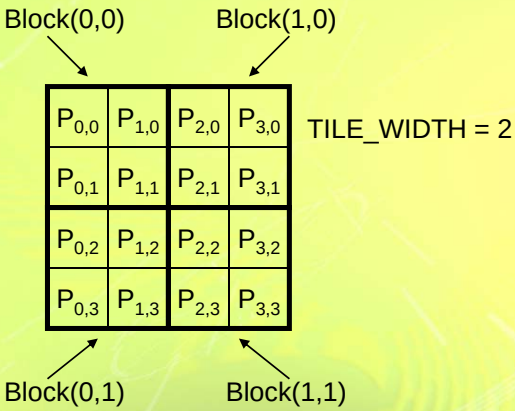
Matrix Multiplication Using Multiple Blocks



Matrix Multiplication Using Multiple Blocks (Cont.)

- tx: threaded.x
- ty: threaded.y
- bx: blockId.x
- by: blockId.y
- Each thread still calculates one Pd element.
- a thread can find the x and y indexes of its Pd element as:
- X index: (bx*tile_width + tx)
- Y index: (by*tile_width + ty)

Example:



Example

Pd element calculated by thread(0,1) of block(1,1)?

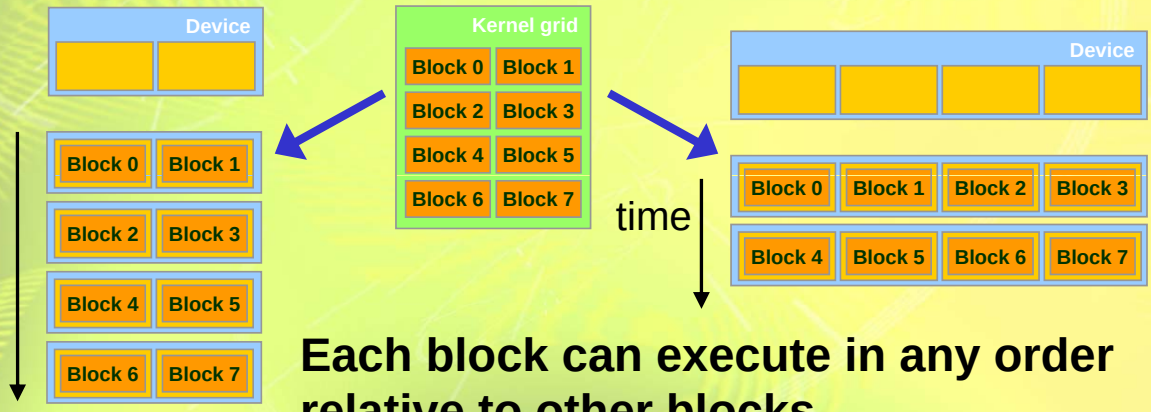
The answer is:

Pd[bx*tile_width+tx][by*tile_width+ty] = Pd[1*2+0][1*2+1] = Pd[2][3].

Revised Matrix Multiplication Kernel using multiple blocks

```
1.  global__ void MatrixMulKernel(float* Md, float* Nd, float* Pd,
    intWidth)
2.  {
3.  // Calculate the row index of the Pd element and M
4.  int Row = blockId.y * TILE_WIDTH + threadId.y;
5.  // Calculate the column idenx of Pd and N
6.  Int Col = blockId.x * TILE_WIDTH + threadId.x;
7.  Pvalue = 0;
8.  // each thread computes one element of the block sub-matrix
9.  for (int k = 0; k < Width; ++k)
10. Pvalue += Md[Row][k] * Nd[k][Col];
11. Pd[Row][Col] = Pvalue;
12. }
```

Transparent Scalability

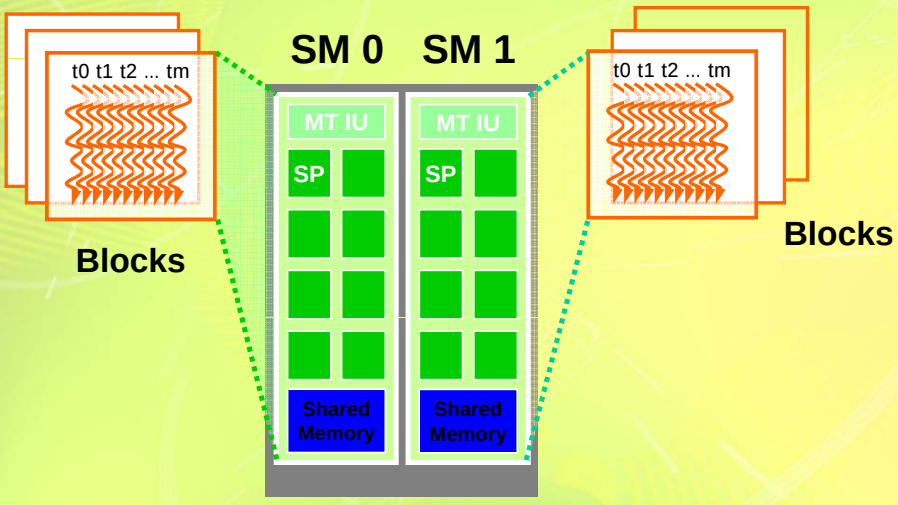


Each block can execute in any order relative to other blocks.

Transparent Scalability (Cont.)

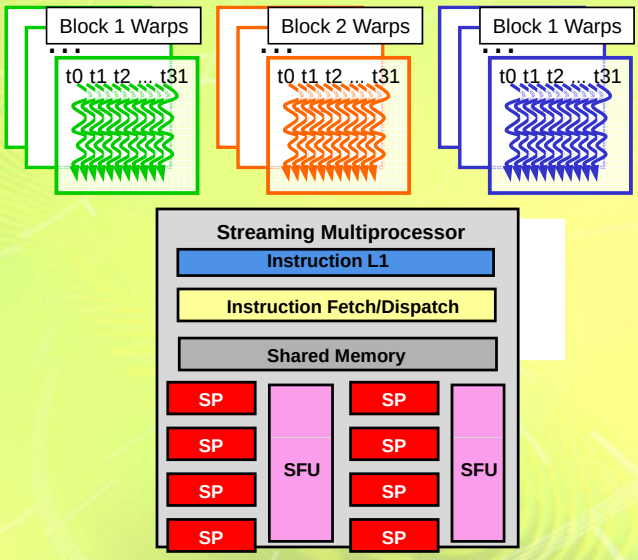
- syncthreads() is a barrier synchronization function that allows the threads in the same block to coordinate their activities.
- When a kernel calls syncthreads() function, all threads of the same block will be stop, until another threads of the block reaches the sync location.
- CUDA has no barrier for synchronize the blocks of a grid, thus blocks can execute in any order relative to each other.

Thread Assignment



Each block is assigned to a SM (Streaming Multiprocessor)

Thread Scheduling



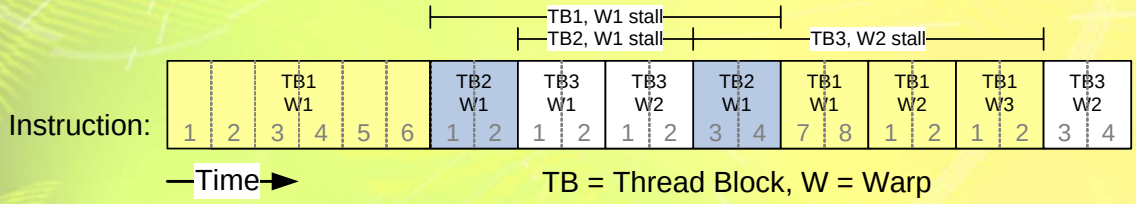
Warp-Based thread scheduling in GeForce 8800GTX

Thread Scheduling: (Cont.)

- **When a block is assigned to a SM, it is further divided into 32-thread units called Warps**
- **Warps are scheduling units in SM**
- **If 8 blocks are assigned to an SM and each block has 96 threads, how many Warps are there in an SM?**
  - **Each Block would have 96/32 = 3 Warps**
  - **There are 8 * 3 = 24 Warps**

Thread Scheduling (Cont.)

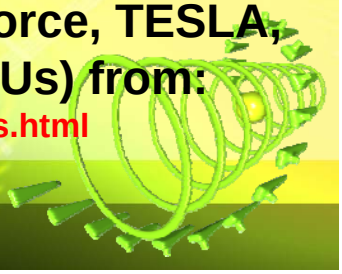
Timing example of warp-based thread scheduling



SM implements zero-overhead warp scheduling

Getting started:

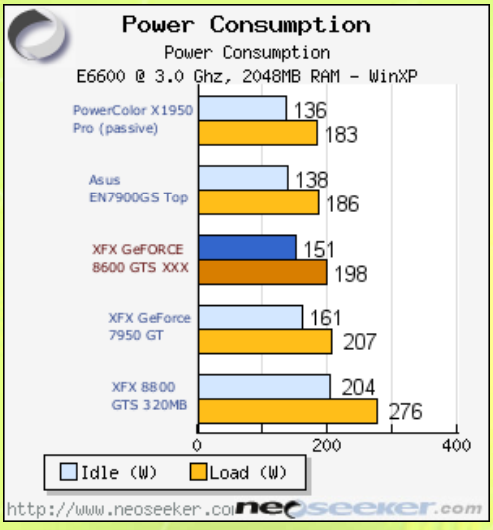
- **First you need to purchase a GeForce 8, 9, 100, 200-series, GPUs with a minimum of 256MB of local graphics memory. For example:**
- **GeForce 8100 mGPU**
- **GeForce 8600 GTS (Under 100$)**
- **GeForce 8800 GTX**
- **GeForce 9600 GT (Under 170$)**
- **You can see more examples of GeForce, TESLA, QUADRO (Nvidia CUDA Support GPUs) from:**
  http://www.nvidia.com/object/cuda_learn_products.html



Getting started: (Cont.)

- For example you can buy a card that uses the NVIDIA GeForce 8600 GPU and has 32 processing elements.
- It is a lot cheaper and smaller than a card with a full-blown 128-processor 8800 GPU.
- Size is important because you have to fit the card into your case!

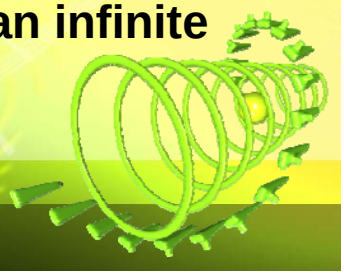
GeFORCE 8600 Power Consumption:



It also uses less power and you can use it with a small power supply.

Install the card:

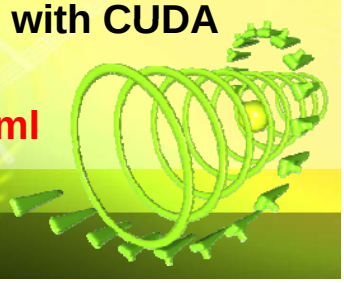
Install the card in the PC, but you don't bother to load any of the packaged drivers because you would be using it as a computational co-processor without any display attached. (You can use it as a combined coprocessor/display card, but the CUDA release notes say that CUDA-related run times would be limited to less than five seconds. I presume that's to prevent losing control of your display if you enter an infinite loop.)



CUDA Toolkit, SDK & Driver:

- Download and install the free Visual C++ 2005 Express Edition to use with CUDA from:
  http://msdn2.microsoft.com/en-us/vstudio/aa700736.aspx

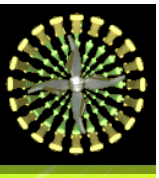
- download three things from the CUDA website:
  * CUDA Toolkit version 1.1 for Windows XP (x86 version)
  * CUDA SDK version 1.1 for Windows XP (x86 version)
  * NVIDIA Driver for Microsoft Windows XP with CUDA Support (169.21, x86 version)
  http://www.nvidia.com/object/cuda_get.html



CUDA Toolkit, SDK & Driver: (Cont.)

- The CUDA Toolkit has all the programming tools, such as the CUDA compiler (nvcc).
- The SDK contains a bunch of CUDA example programs.
- install both into their default directories. Then install the NVIDIA driver and restarted your PC.

Recompile and run:

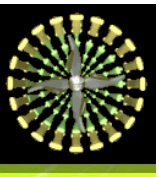


- Now recompile and run the project found at:

  C:\Program Files\NVIDIA Corporation\NVIDIA CUDA SDK\projects\deviceQuery
- This project just queries the system and looks for any CUDA-enabled devices that exist.
- Double-click the deviceQuery.sln project file and the project appear in your Visual C++ 2005 IDE.
- select the Debug configuration for Win32 and rebuild it. Then execute the program in the debugger. A console window appears that says:

  Cuda error in file 'deviceQuery.cu' in line 53 : initialization error.

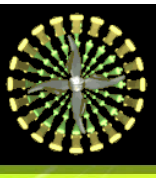
emuDebug configuration:



- **Why dose this message appear?**
- **You should select the emuDebug configuration, build it and run it.**
- **This configuration uses a software emulation of a CUDA device instead of the actual hardware found on the graphics card. No errors this time and you get a list of the characteristics of the emulated CUDA device, but the first line of the console output is:**

  There is no device supporting CUDA.

Disabling your virus protection



You know the graphic card is in the PC, and device manager reports that the NVIDIA GeForce 8600 GTS display adapter was working properly. You can find some hints on installing the NVIDIA driver, form:

http://www.nvidia.com/object/driver_installation_hints.html

and one of the things you may haven't done is disabling your virus protection during driver installation.

Driver Installation Hints

Driver Installation Hints:

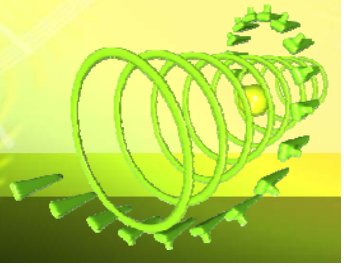
- **"Download Accelerator" utilities should be disabled when downloading any drivers.**
- **Do not run virus protection software in the background while installing the drivers. This prevents the driver from configuring itself properly.**

Rebuild and re-run:

So uninstall the NVIDIA driver, disable your virus scanner, reinstall the driver, restart your PC, reopen the deviceQuery project, rebuild the Debug configuration, re-run it and get exactly the same error!

CUDA not detecting your graphic card when it is not the primary display driver!

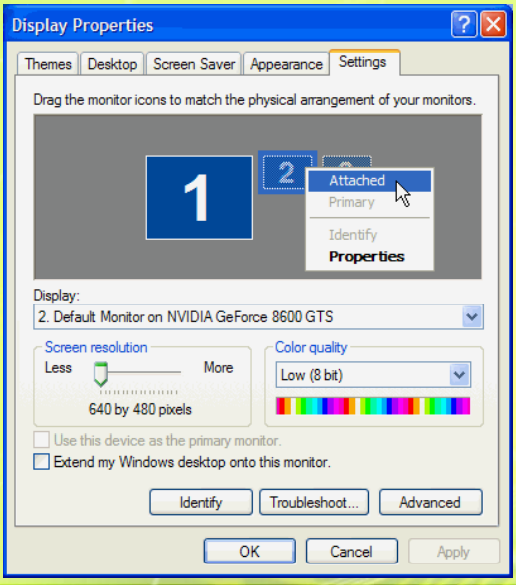
So, what is the problem?



What is the problem?

- **Right-click on your desktop, open the Display Properties window and notice that the GeForce 8600 GTS is not attached to any display!**
- **What if you attach it, even though there is no physical display connected to the card?**
- **right-click on the display icon for the 8600 and select Attached, after which click on the OK button for the Display Properties window.**

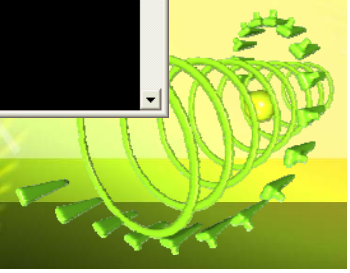
Attach GeForce 8600 GTS to a display:



Re-run:

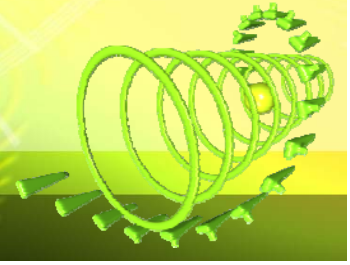
Then re-run the deviceQuery program and get the following result:

```
C:\WINDOWS\system32\cmd.exe
There is 1 device supporting CUDA

Device 0: "GeForce 8600 GTS"
  Major revision number:                         1
  Minor revision number:                         1
  Total amount of global memory:                 268173312 bytes
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       16384 bytes
  Total number of registers available per block: 8192
  Warp size:                                     32
  Maximum number of threads per block:           512
  Maximum sizes of each dimension of a block:    512 x 512 x 64
  Maximum sizes of each dimension of a grid:     65535 x 65535 x 1
  Maximum memory pitch:                          262144 bytes
  Texture alignment:                             256 bytes
  Clock rate:                                    1458000 kilohertz

Test PASSED

Press ENTER to exit...
```



Congradulations

So, now you're ready to begin using CUDA.



Web Resources

- **CUDA ZONE: http://www.nvidia.com/object/cuda_education.html**
- ***David Kirk and Wen-mei W. Hwu,* Lecture Notes of Programming Massively Parallel Processors, University of Illinois, Urbana-Champaign, *2009***
- **Rob Farber, "CUDA, Supercomputing for the Masses", *Dr. Dobb's Journal, can be found at:* http://www.ddj.com/hpc-high-performance-computing/**

Acknowledgment

- **Special Thanks to Mr. Ebrahim Khademi for his technical comments.**


Any Questions?
Any Question?