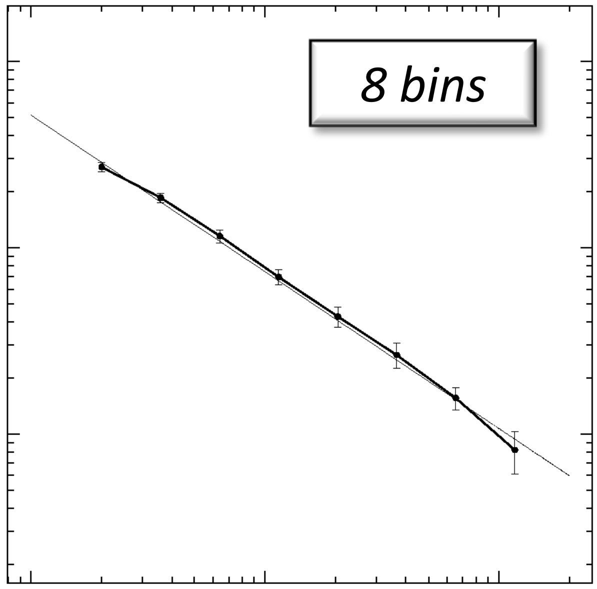
Graphics Processor Programming in CUDA

7/18/2012

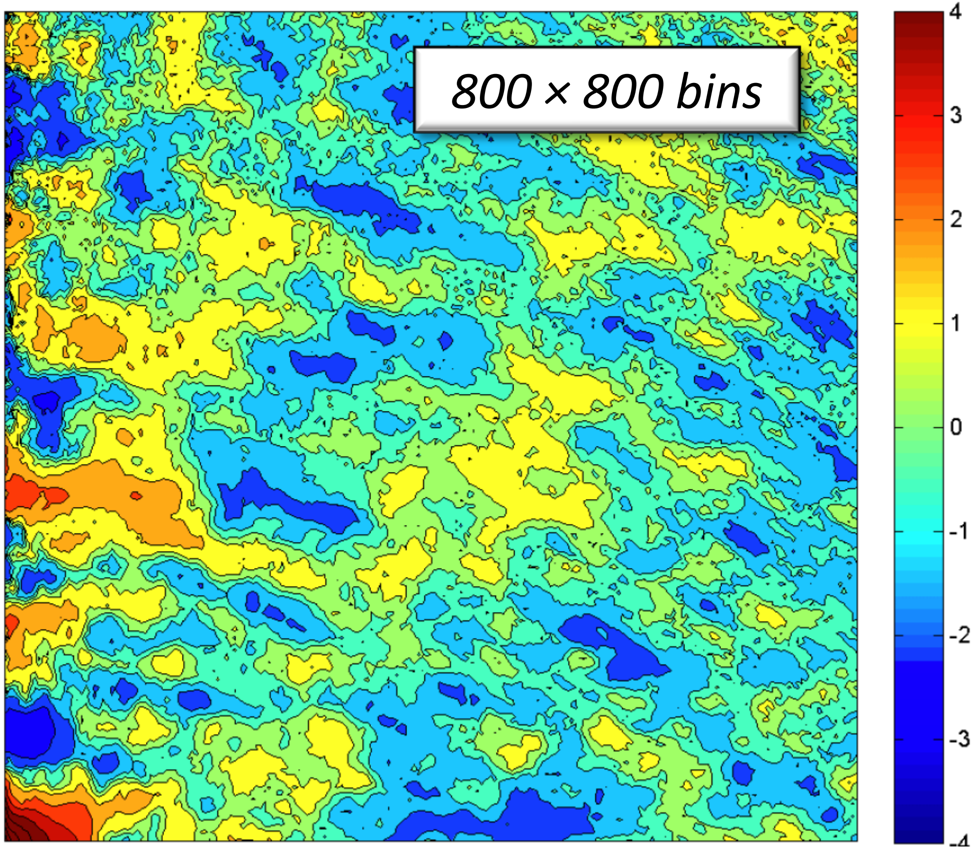
Tamás Budavári / The Johns Hopkins University

How I got into this?

- Galaxy correlation function
  - Histogram of distances
- State-of-the-art method
  - Dual-tree traversal

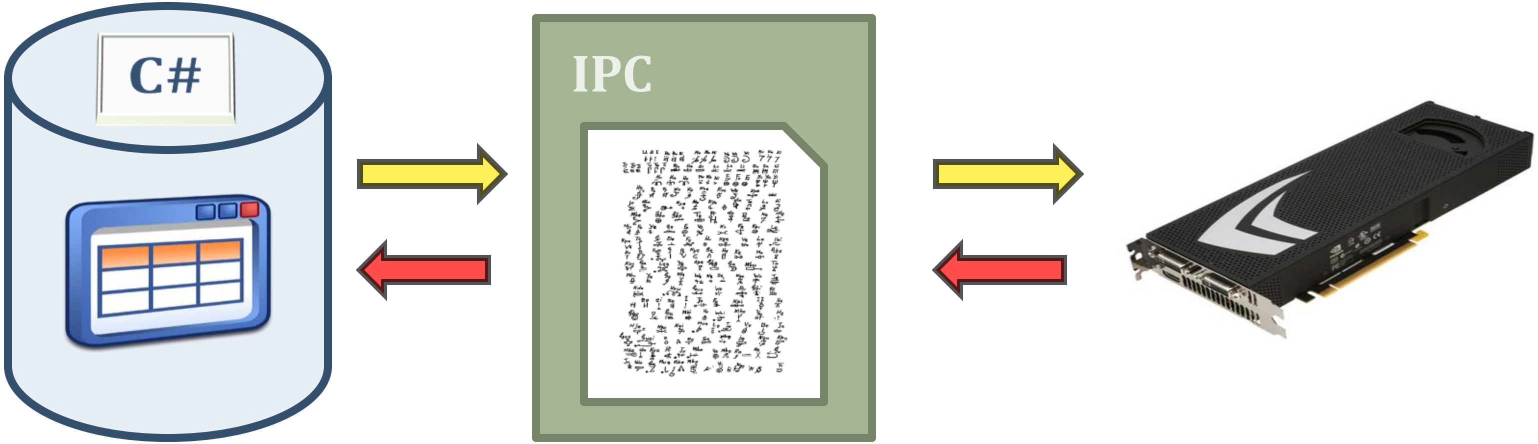


What if?



Extending SQL Server

- Dedicated service for direct access
  - Shared memory IPC w/ on-the-fly data transform



User-Defined Functions

- Pair counts computed on the GPU
  - Returns 2D histogram as a table (i, j, cts)
- Calculate the correlation fn in SQL

```
dbo.PairCounts(@maxmpc, @nbin, @qryD, @nD, null) dd
```

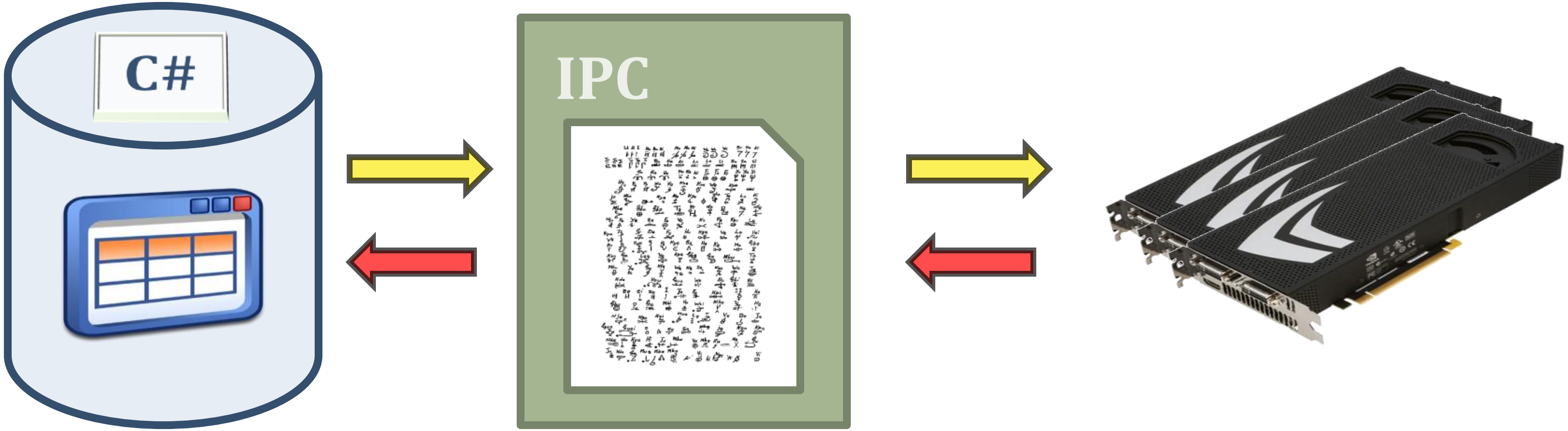
User-Defined Functions

- Pair counts computed on the GPU
  - Returns 2D histogram as a table (i, j, cts)
- Calculate the correlation fn in SQL

```
select dd.i, dd.j,  dd.cts as dd, dr.cts as dr, rr.cts as rr
    (@Nrr*CONVERT(float,dd.cts)/@Ndd - 2*@Nrr*CONVERT(float,dr.cts)/@Ndr + CONVERT(float,rr.cts))
        / CONVERT(float,rr.cts) as xi
from     dbo.PairCounts(@maxmpc, @nbin, @qryD, @nD, null) dd
    join dbo.PairCounts(@maxmpc, @nbin, @qryR, @nR, null) rr  on dd.i = rr.i and dd.j = rr.j
    join dbo.PairCounts(@maxmpc, @nbin, @qryDR, @nD, @nR) dr  on dd.i = dr.i and dd.j = dr.j
go
```

Multiple GPUs in Parallel

- Several C# proxies to launch jobs on more cards
  - Non-blocking SQL routines



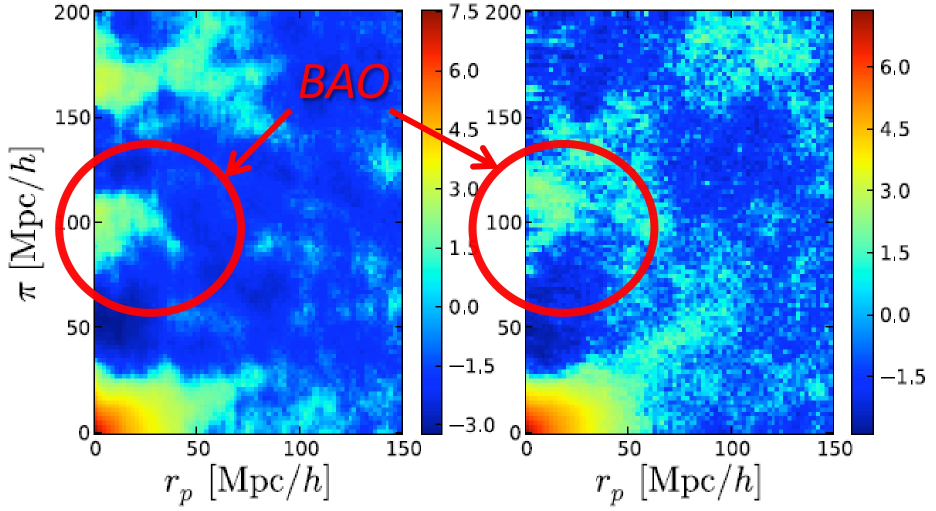
Async SQL Interface


```sql
declare @dd_req uniqueidentifier, @rr_req uniqueidentifier, @dr_req uniqueidentifier;

-- launch jobs on GPUs 0,1,2
set @dd_req = dbo.PairCountsAsyncBegin(0, @timeout, @maxmpc, @nbin, @qryD, @nD, null);
set @rr_req = dbo.PairCountsAsyncBegin(1, @timeout, @maxmpc, @nbin, @qryR, @nR, null);
set @dr_req = dbo.PairCountsAsyncBegin(2, @timeout, @maxmpc, @nbin, @qryDR, @nD, @nR);

-- wait for results
select dd.i, dd.j,  dd.cts as dd,  dr.cts as dr,  rr.cts as rr, ...
from     dbo.PairCountsAsyncEnd(@dd_req, @timeout) dd
    join dbo.PairCountsAsyncEnd(@rr_req, @timeout) rr  on dd.i = rr.i and dd.j = rr.j
    join dbo.PairCountsAsyncEnd(@dr_req, @timeout) dr  on dd.i = dr.i and dd.j = dr.j
go
```

Baryon Acoustic Oscillations

- 600 trillion galaxy pairs
- C for CUDA on GPUs

Tian, Neyrinck, TB & Szalay (2011)



Outline

- Parallelism
- Hardware
- Programming
- Multithreading
- Coding for GPUs
- CUDA, Thrust, …

Parallelism

- Data parallel
  - Same processing on different pieces of data
- Task parallel
  - Simultaneous processing on the same data

On all levels of the hierarchy

- Clouds
- Clusters
- Machines
- Cores
- Threads

Scalability

- Scale up
  - Vertically
  - Add resources to a node
    - Bigger memory, …
    - Faster processor, …

- Scale out
  - Horizontally
  - Use more of the
    - Threads, cores, machines, clusters, clouds, …

Cluster

High-Performance Computing

- Traditional HPC clusters
  - Launching jobs on a cluster of machines
  - Use MPI to communicate among nodes
    - Message Passing Interface

Queuing Systems

- Used for batch jobs on computer clusters
  - Fair scheduling of user jobs
  - Group policies
- Several systems
  - Portable Batch System (PBS)
  - Condor, etc…

Computer

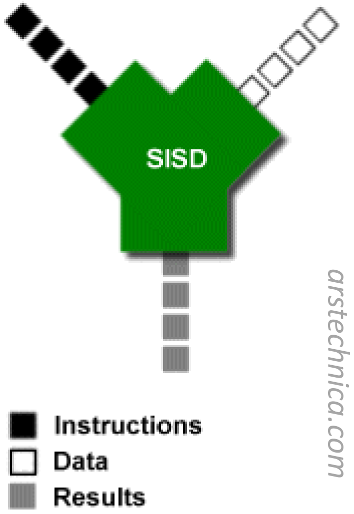
Classification of Parallel Computers

- Flynn's Taxonomy

	Single instruction	Multiple instruction
Single data	SISD	MISD
Multiple data	SIMD	MIMD

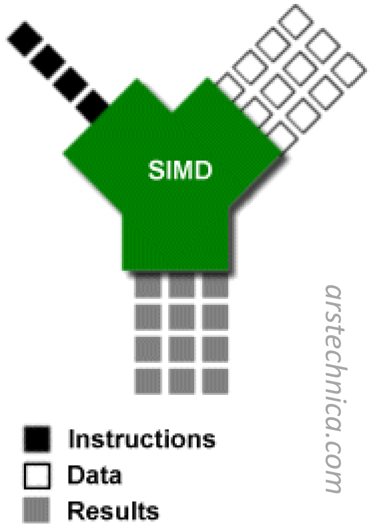
SISD

- Single Instruction Single Data
  - Classical Von Neumann machines
  - Single threaded codes



SIMD

- Single Instruction Multiple Data
  - On x86
    - MMX: Math Matrix eXtension
    - SSE: Streaming SIMD Extension
    - …and more…
  - GPU programming!!



Amdahl's Law of Parallelism

- Speed up:

$$\frac{T(1)}{T(N)} = \frac{S+P}{S+\frac{P}{N}}$$

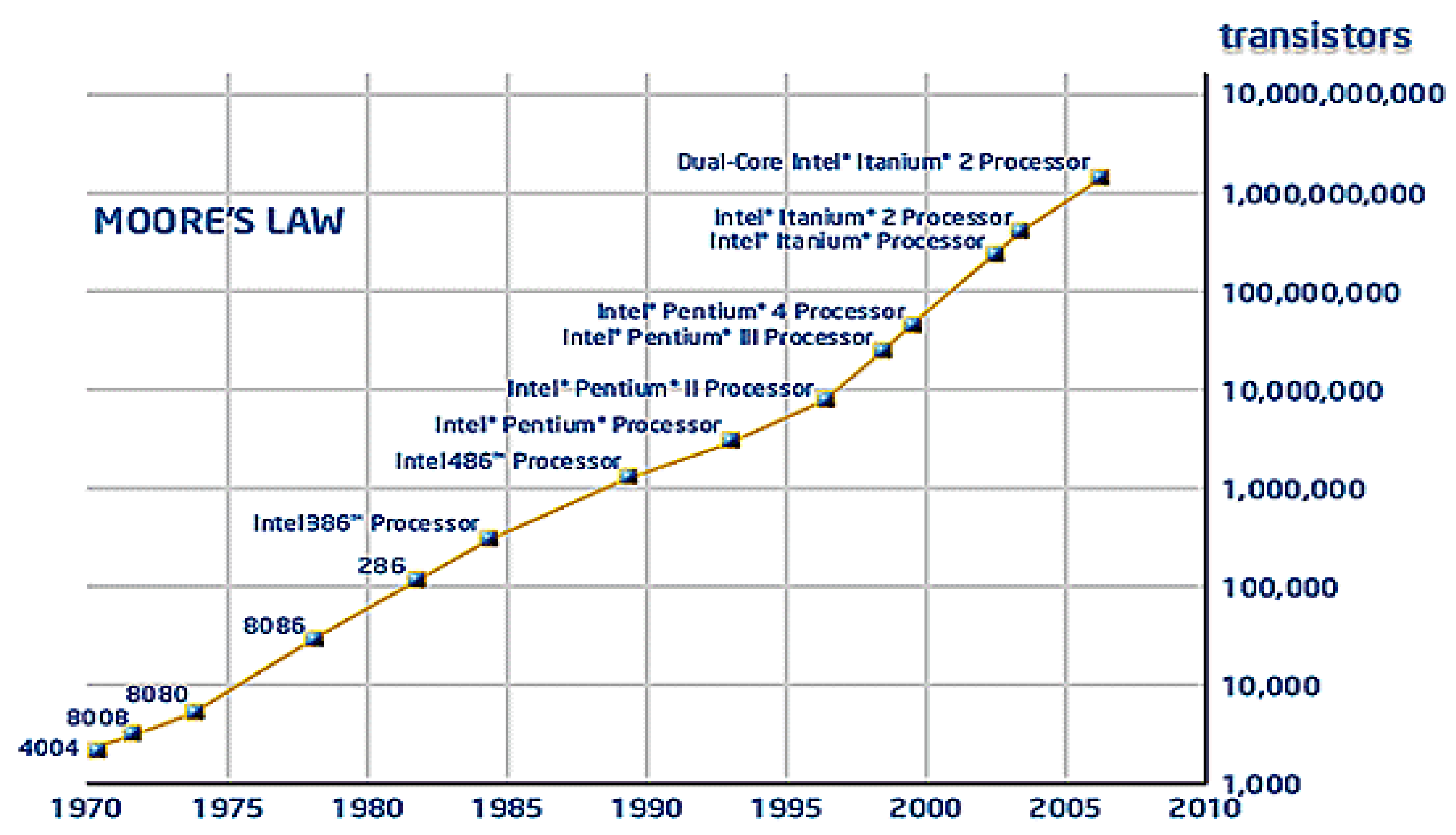
with $$p = \frac{P}{S+P}$$

$$\frac{T(1)}{T(N)} = \frac{1}{(1-p)+\frac{p}{N}}$$

  - Before looking into parallelism, speed up the serial code, to figure out the max speedup, i.e., $N \to \infty$

Chip

Moore's Law


transistors
MOORE'S LAW
10,000,000,000
1,000,000,000
100,000,000
10,000,000
1,000,000
100,000
10,000
1,000
1970
1975
1980
1985
1990
1995
2000
2005
2010
Dual-Core Intel* Itanium* 2 Processor
Intel* Itanium* 2 Processor
Intel* Itanium* Processor
Intel* Pentium* 4 Processor
Intel* Pentium* III Processor
Intel* Pentium* II Processor
Intel* Pentium* Processor
Intel486™ Processor
Intel386™ Processor
286
8086
8080
8008
4004

New Limitation is Energy!

- Power to compute the same thing?
  - CPU is 10× less efficient than a digital signal processor
  - DSP is 10× less efficient than a custom chip
- New design: multicores with slower clocks
  - But the interconnect is expensive
  - Need simpler components

Emerging Architectures

- Andrew Chien: 10×10 to replace the 90/10 rule
  - Custom modules on chip, cf. SoC in cellphones

Emerging Architectures

- Andrew Chien: 10×10 to replace the 90/10 rule
  - Custom modules on chip, cf. SoC in cellphones

- Scientific analysis on such specialized units?

GPUs Evolved to be General Purpose

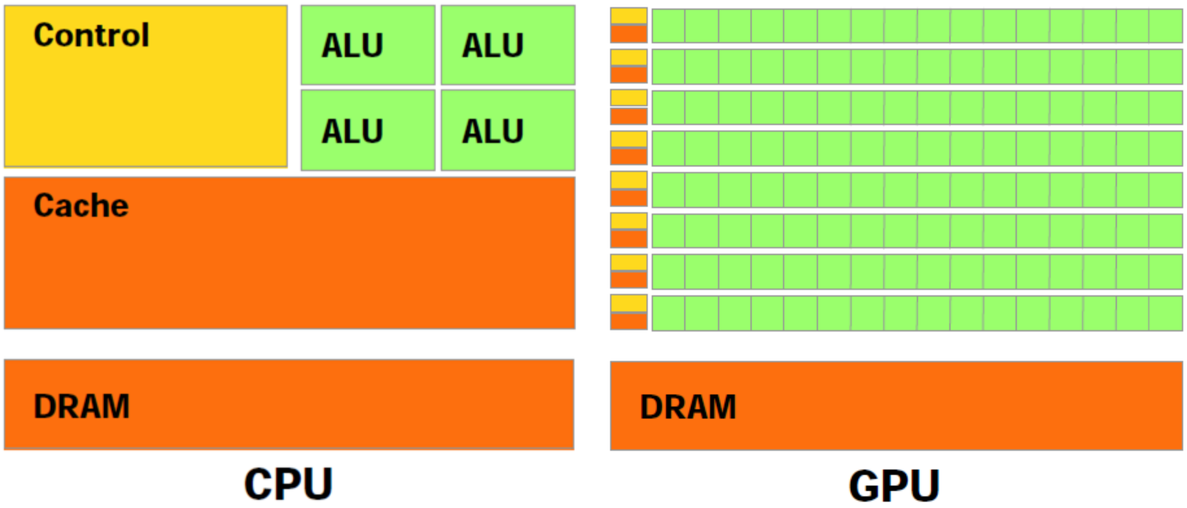
- Virtual world: simulation of real physics
  - C for CUDA and OpenCL

- 512 cores ~25k threads, running 1 billion/sec
- Old algorithms built on wrong assumption
  - Today processing is free but memory is slow

New programming paradigm!

New Moore's Law

- In the number of cores
- Faster than ever



Programming

Programming Languages

- No one language to rule them all
- And many to choose from

Assembly

- Low-level (almost) machine code
- Different for each computer

Reserved for life-threatening situations!

The “C” Language

- Higher level but still close to hardware, i.e., fast
  - Pointers!
- Many things written in C
  - Operating systems
  - Other languages, …

Java

- Pros
  - Memory management with garbage collection
  - Just-In-Time compilation from ‘bytecode’
- Cons
  - Not so great performance
  - Hard to include legacy codes
  - New language features were an afterthought

Python

- Scripting to glue things together
- Easy to wrap legacy codes
- Lots of scientific modules and plotting
- Good for prototyping

Etc…

- Perl
- Matlab
- Mathematica
- IDL
- R
- Lisp
- Haskell
- Ocaml
- Erlang
- Your favorite here…

Programming in C

- Skeleton of an application

```c
#include <stdio.h>
#include <math.h>

int main(int argc, char* argv)
{
    float x = 0;
    float c = cos(x);
    printf("cos(0) = %f\n", c);
    return 0;
}
```

Programming in C

- Files
  - Headers *.h
  - Source *.c
- Building an application
  - Compile source
  - Link object files

Using Pointers


```c
#include <stdio.h>
#include <math.h>

int main(int argc, char* argv)
{
    float x, y;  // float values
    float* p;    // pointer to a float
    x = 0;       // assignment
    p = &x;      // fetch pointer to x
    x = cos(x);  // update value of x
    y = *p;      // value where p points
    printf("y = %f\n", y);
    return 0;
}
```

Arrays

- Dynamic arrays
  - Memory allocation
  - Freeing memory
- Pointer arithmetics

```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char* argv)
{
    int num = 3; // number of vector elements

    // allocate memory
    float* p = (float*) malloc(num * sizeof(float));
    if (p == NULL) return 1; // quit with error

    // load data
    for (int i=0; i<num; i++) p[i] = i*i;

    // pointer arithmetics: 'p[i]' means '*(p+i)'
    float* q = p + num-1;
    printf("first element: %f\n", *p);
    printf("last  element: %f\n", *q);

    // release allocated memory and quit w/o error
    free(p);
    return 0;
}
```

Matrix, etc…

- Point to pointers
  - Data allocated in **v**
  - Pointers in **A**
    - For 2D indexing
- One can have
  - Matrix, tensor, …
  - Jagged arrays, …

```c
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char* argv)
{
    int N=3, M=2;

    // allocate contiguous memory for data
    float* v = (float*) malloc(N*M * sizeof(float));
    if (v == NULL) return 1; // quit with error 1

    // allocate array of pointers
    float** A = (float**) malloc(N * sizeof(float*));
    if (A == NULL) return 2; // quit with error 2

    // set up pointers for matrix
    for (int i=0; i<N; i++) A[i] = v + i * M;

    // load some data
    for (int k=0; k<N*M; k++) v[k] = k;

    // print matrix
    for (int i=0; i<N; i++)
    {
        for (int j=0; j<M; j++) printf(" %f", A[i][j]);
        printf("\n");
    }
    free(A); free(v);
    return 0;
}
```

Concurrency

Parallel actions

Data Parallel Techniques

- "Embarrassingly Parallel"
  - Decoupled problems, independent processing

- MapReduce
  - Map
  - Reduce

The Elevator Problem

- People on multiple levels
  - Press the button…

Mutual Exclusion

- Multiple processes or threads
  - Access shared resources in critical sections
    - E.g., call the elevator when it's time to go

- Locking
  - Elevators, etc…

Dining Philosophers

- Five silent philosophers sit at the table
  - Alternate between eating and thinking
  - Need both forks left & right to eat
    - Must be picked up one by one!
  - Infinite food in front of them
- How can they all think & eat forever?

Image from Wikipedia

Parallel Threads

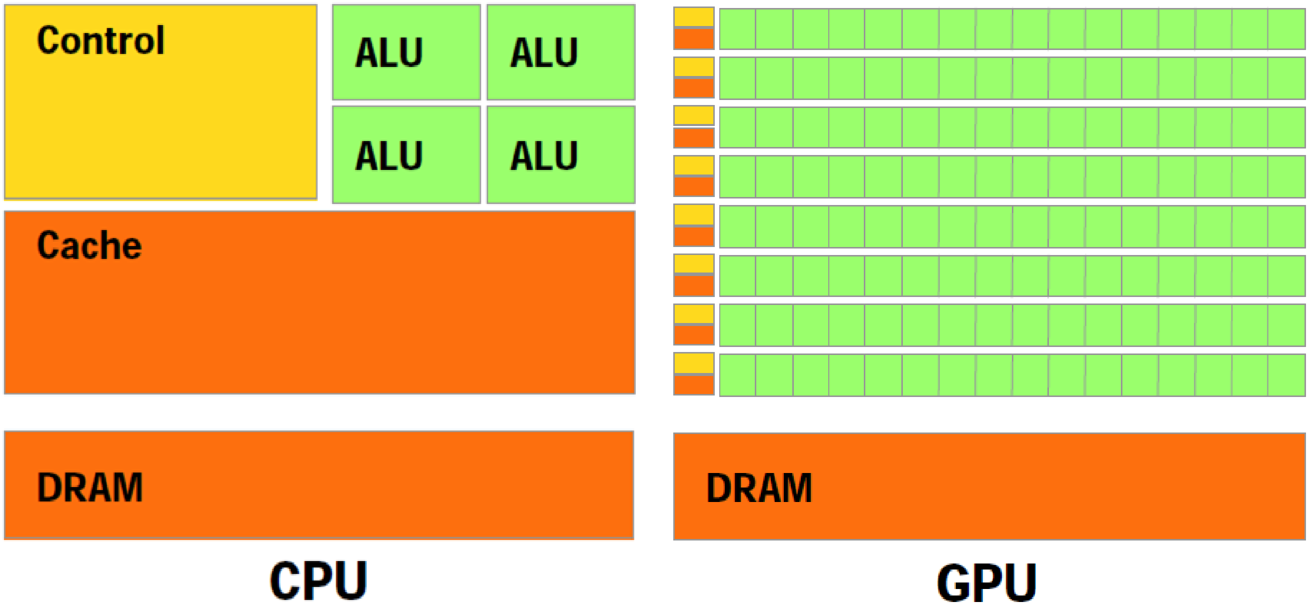
Threading

- Concurrent parallelism in a machine

Parallelism

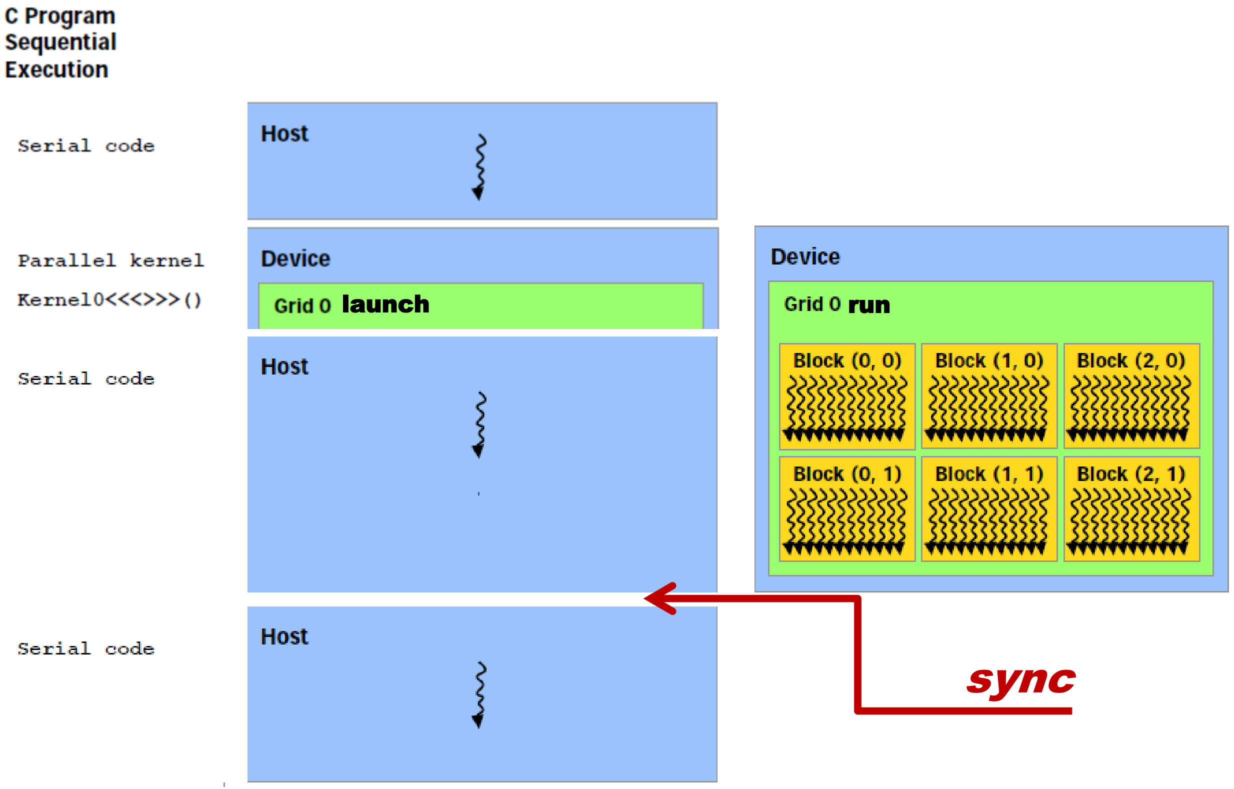
- Data parallel
  - Same processing on different pieces of data
- Task parallel
  - Simultaneous processing on the same data

Comparing Chips



Hybrid Architecture

Tamás Budavári

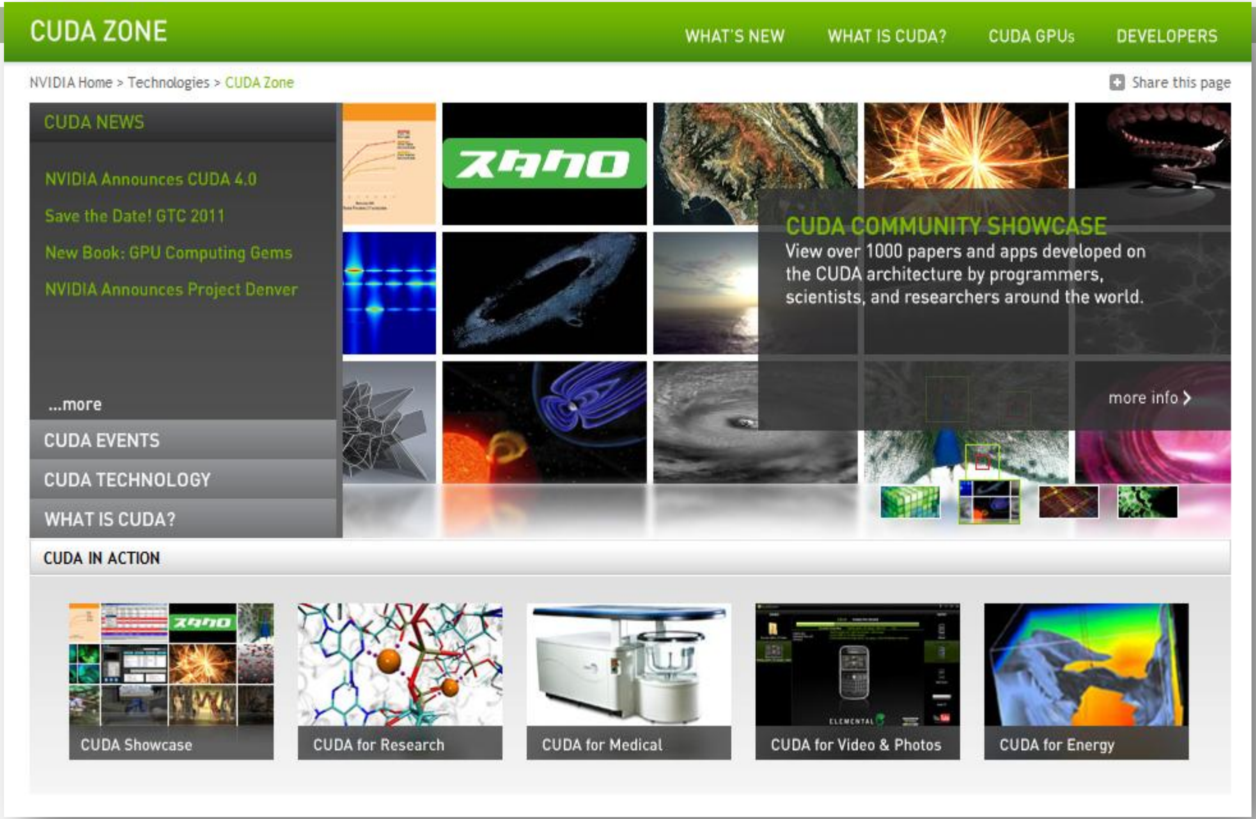


Programming GPGPUs

- CUDA
  - Low-level & high-level
- OpenCL
- DirectCompute
  - DirectX, etc…
- C++ AMP *New!*
  - Accelerated Massive Parallelism

CUDA

Projects on CUDA Zone



Currently Available

Tamás Budavári

- GPU optimized Sorting, RNG, BLAS, FFT, Hadamard...
- SDK w/examples
- Nsight debugger!

- Imaging routines
- Python w/ PyCUDA
- High-level C++ programming with

Fermi

- Previous generation
  - 20 series Tesla cards, e.g., C2050
  - 400+ series GeForce cards, e.g., GTX 480

- IEEE-754 arithmetic
  - Standard floating point
    - Same as in the CPUs

Kepler

- Latest generation
  - More efficient, more cores
    - GTX 680 has 1536 cores

Which Device?

- Tesla
  - Computation (& games)
  - Up to 6GB memory
  - ECC on/off
    - Error Correcting Codes
  - More double-precision units on chip
- GeForce
  - Games (& computation)
  - Typically 1.5GB memory
  - Faster CLK & more cores
    - Heat! Not to run for years
  - Great for development and more…

Multiprocessors

Tamás Budavári

- 1536 simultaneous threads
  - Up to ~25,000 threads on a Fermi GPU w/16 MPs
- L1 cache memory – implicitly speeds codes up
- Shared memory – explicit programming
  - 100× access speed compared to global memory
  - Allows for fast communication between threads
- L1 + Shared is 64KB (same chip)
  - Configurable to 16KB + 48KB or 48KB+16KB

Kernel

- GPU code to run on all threads
  - Pick your data
  - Process it
  - Save the results

Warp

- Threads are grouped into warps
  - 32 threads running together SIMD style

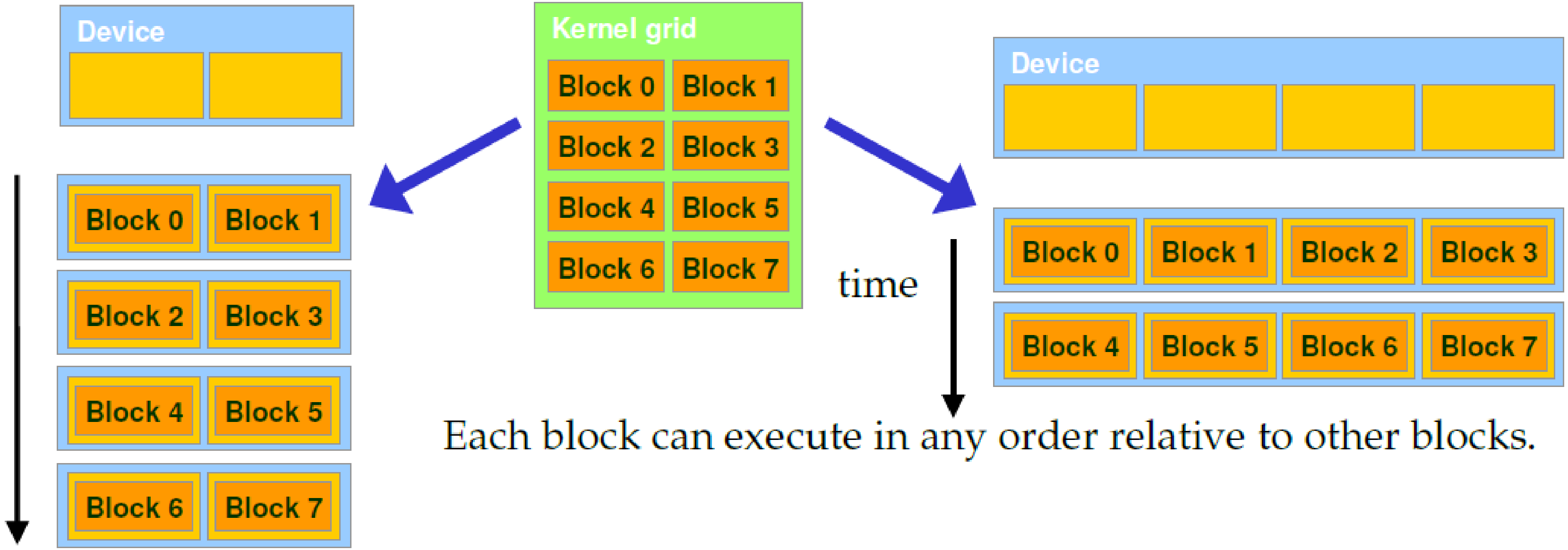
Block

- Block of threads
  - 1D, 2D or 3D to best match the data layout
  - Can communicate with each other!

Grid

- Grid of blocks
  - Launch millions of threads
    - Regardless how many cores available
  - No communication
    - Different blocks can be running sequentially or different processor

Same code on different devices



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

Hello World!


```
int main()
{
    ...
    // Kernel invocation
    VecAdd<<<1, N>>>(A, B, C);
}
```

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}
```

To Be Continued…