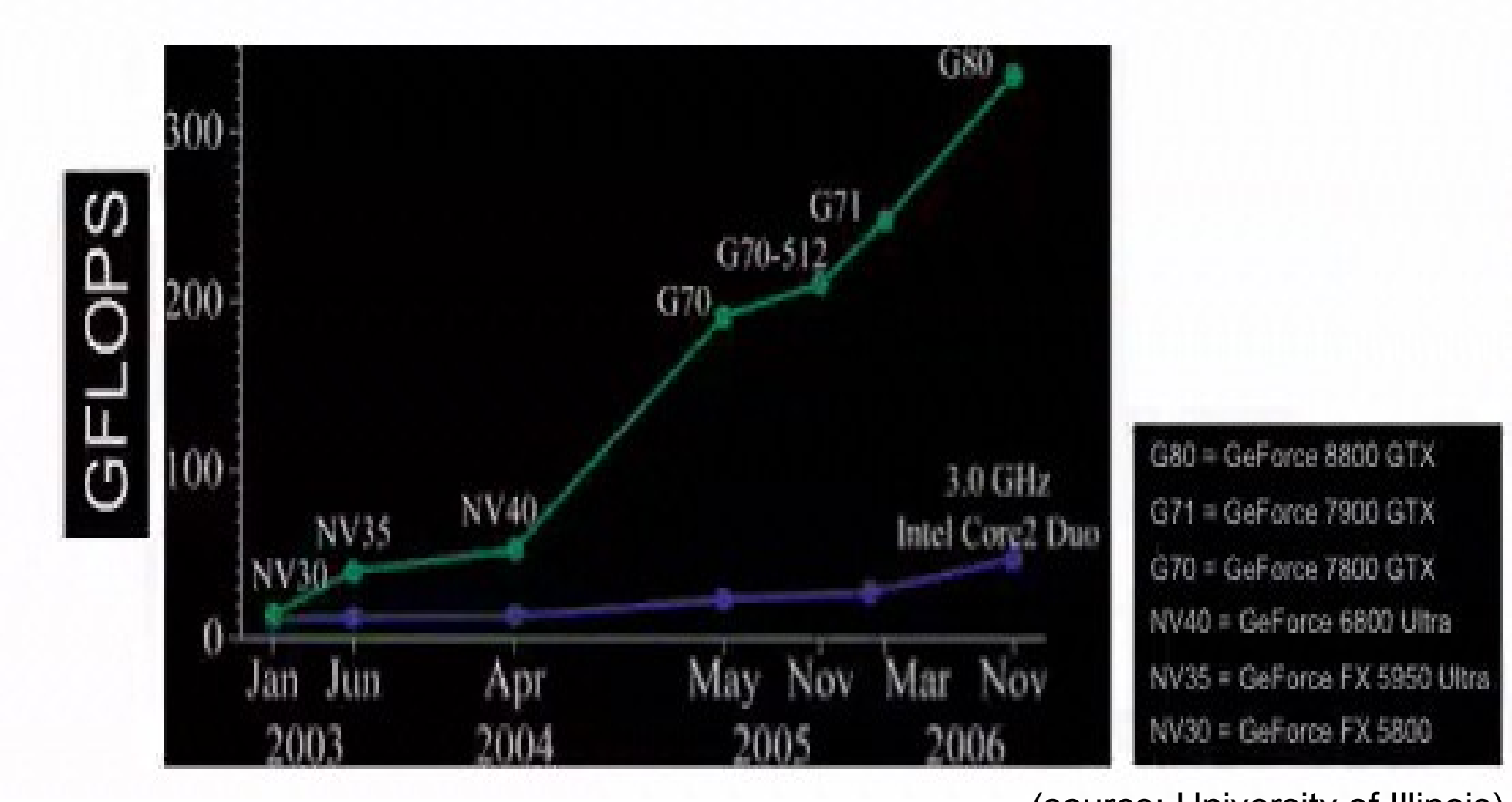
CUDA

Nvidia Grafikkarten unter Linux anzapfen

(Hans-Jürgen Schönig)

Wozu GPU Programmierung?



(source: University of Illinois)

CPU vs. GPU

- CPUs sind ...
  - flexibel einsetzbar
  - für sequentielle Operationen optimiert
  - guter “Integer” Support

- GPUs sind ...
  - hochgradig parallel
  - einzelne “Threads” sind langsamer
  - auf Floating Point optimiert

GPU Hardware

```
$ ./deviceQuery
There are 2 devices supporting CUDA

Device 0: "GeForce 9800 GT"
 Major revision number:                         1
 Minor revision number:                         1
 Total amount of global memory:                 536150016 bytes
 Number of multiprocessors:                     14
 Number of cores:                               112
 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:                                    1.73 GHz
 Concurrent copy and execution:                 Yes
```

Cuda Grundkonzept

- Cuda bietet kleine Erweiterungen für C und C++

- XX.XXX gleichzeitige Threads sind kein Problem

- Die Hardware scheduled die Ausführung

- Threads können in “Thread Blocks” organisiert werden

- “Thread Blocks” können zu Grids zusammengefasst werden.

Daten kopieren

```
/* implement random generator and copy to CUDA */
nn_precision*
generate_random_numbers(int number_of_values)
{
	nn_precision	*cuda_float_p;

	/* allocate host memory and CUDA memory */
	nn_precision *host_p = (nn_precision *)pg_palloc(sizeof(nn_precision) * number_of_values);
	CUDATOOLS_SAFE_CALL( cudaMalloc( (void**) &cuda_float_p,
		sizeof(nn_precision) * number_of_values));

	/* create random numbers */
	for	(int i = 0; i < number_of_values; i++)
	{
		host_p[i] = (nn_precision) drand48();
	}

	/* copy data to CUDA and return pointer to CUDA structure */
	CUDATOOLS_SAFE_CALL( cudaMemcpy(cuda_float_p, host_p,
		sizeof(nn_precision) * number_of_values, cudaMemcpyHostToDevice) );

	return cuda_float_p;
}
```

Simple matrix math / CPU

```
void add_matrix ( float* a, float* b, float* c, int N )
{
	int index;
	for ( int i = 0; i < N; ++i )
	{
		for ( int j = 0; j < N; ++j )
		{
			index = i + j*N;
			c[index] = a[index] + b[index];
		}
}

int main()
{
	add_matrix( a, b, c, N );
}
```

Simple matrix math / GPU

```
__global__ add_matrix( float* a, float* b, float* c, int N )
{
	int i = blockIdx.x * blockDim.x + threadIdx.x;
	int j = blockIdx.y * blockDim.y + threadIdx.y;
	int index = i + j*N;
	if ( i < N && j < N )
		c[index] = a[index] + b[index];
}

int main()
{
	dim3 dimBlock( blocksize, blocksize );
	dim3 dimGrid( N/dimBlock.x, N/dimBlock.y );
	add_matrix<<<dimGrid, dimBlock>>>( a, b, c, N );
}
```

Compiling code

As simple as that:

```
$ nvcc --cuda --host-compilation C \
    cuda_tools.cu -o cuda_tools.c
```

- nvcc führt Cuda Code in C / C++ Code über

- Code kann normal mit GCC benutzt werden.

Beispiel: Cuda mit PostgreSQL

```
/* import postgres internal stuff */
#include "postgres.h"
#include "fmgr.h"
#include "funcapi.h"
#include "utils/memutils.h"
#include "utils/elog.h"
#include "cuda_tools.h"

PG_MODULE_MAGIC;

/* prototypes to silence compiler */
extern      Datum      test_random(PG_FUNCTION_ARGS);

/* define function to allocate N random values (0 - 1.0) and put it into the CUDA device */
PG_FUNCTION_INFO_V1(test_random);
Datum
test_random(PG_FUNCTION_ARGS)
{
        int     number = PG_GETARG_INT32(0);
        nn_precision *p = generate_random_numbers(number);
        cuda_free_array(p);

        PG_RETURN_VOID();
}
```

Beispiel: Cuda mit PostgreSQL

```
/* import postgres internal stuff */
#include "postgres.h"
#include "fmgr.h"
#include "funcapi.h"
#include "utils/memutils.h"
#include "utils/elog.h"
#include "cuda_tools.h"

PG_MODULE_MAGIC;

/* prototypes to silence compiler */
extern      Datum      test_random(PG_FUNCTION_ARGS);

/* define function to allocate N random values (0 - 1.0) and put it into the CUDA device */
PG_FUNCTION_INFO_V1(test_random);
Datum
test_random(PG_FUNCTION_ARGS)
{
        int     number = PG_GETARG_INT32(0);
        nn_precision *p = generate_random_numbers(number);
        cuda_free_array(p);

        PG_RETURN_VOID();
}
```

Cuda mit PostgreSQL (2)

- einfache Integration von Cuda möglich

- leicht zu erlernen

- ROLLBACK auf eine GPU ist einfach cool ;)

- Daten haben “kurze Wege”

Any questions?

www.postgresql-support.de