Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

NQueens on CUDA

Frank Feinbube

HPI Research School
Operating Systems and Middleware
Prof. Dr. Andreas Polze

Agenda

- Motivation
- CUDA Overview
- Parallelizing the NQueens Problem
- Optimizations
- Evaluation
- Summary / Conclusion

Agenda

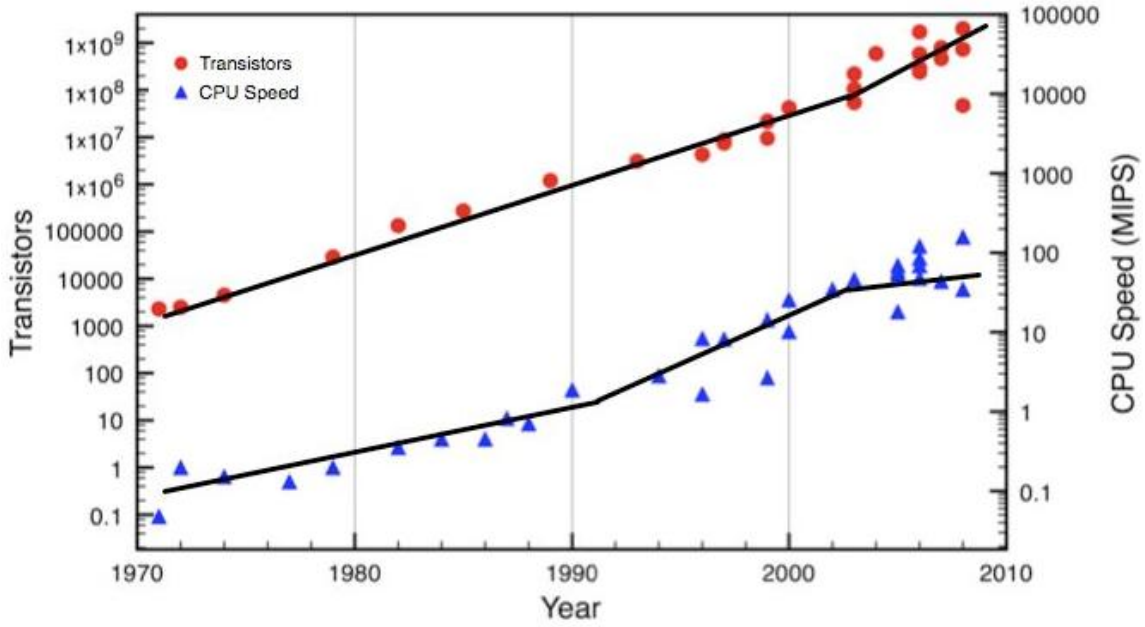
- **Motivation**

- CUDA Overview
- Parallelizing the NQueens Problem
- Optimizations
- Evaluation

- Summary / Conclusion

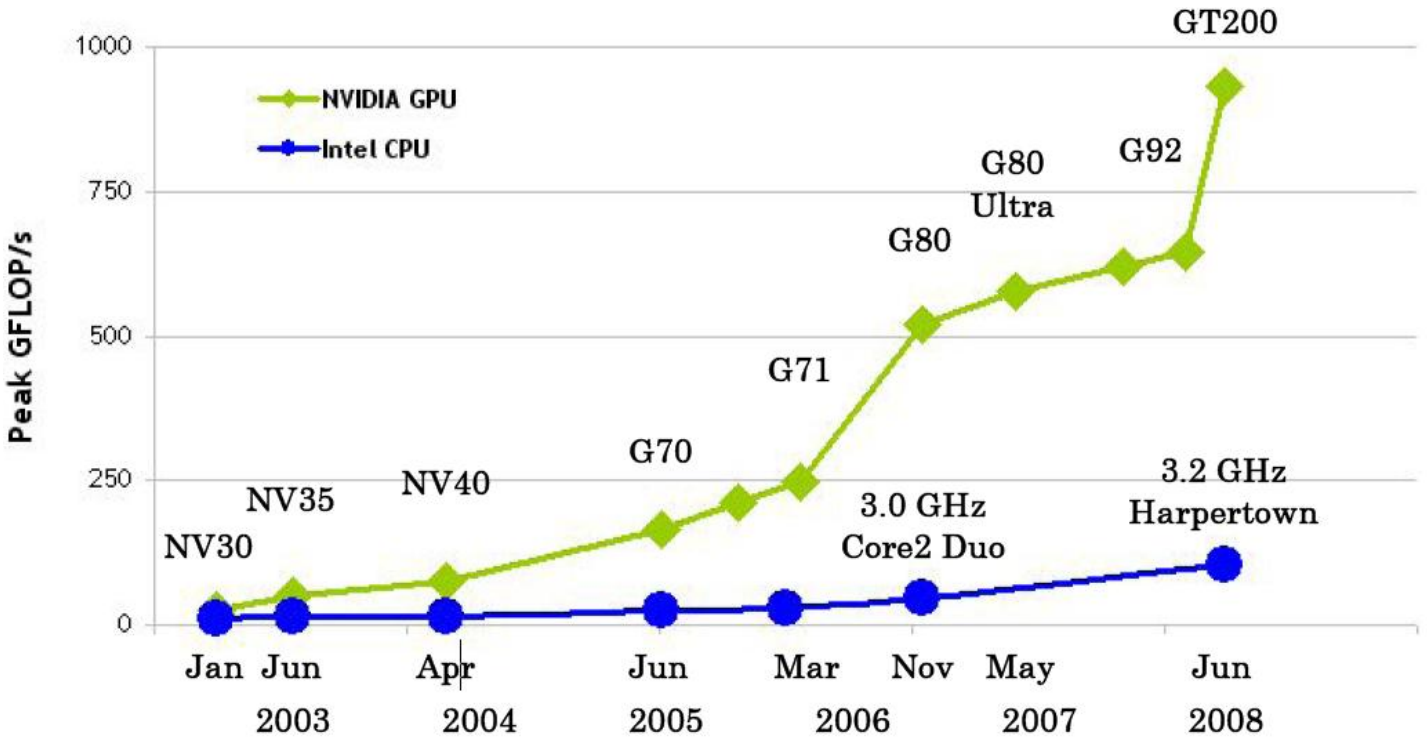
The free lunch is over…

- Moore's law:
  - The number of transistors continues to climb, at least for now
  - Clock speed, however, is a different story



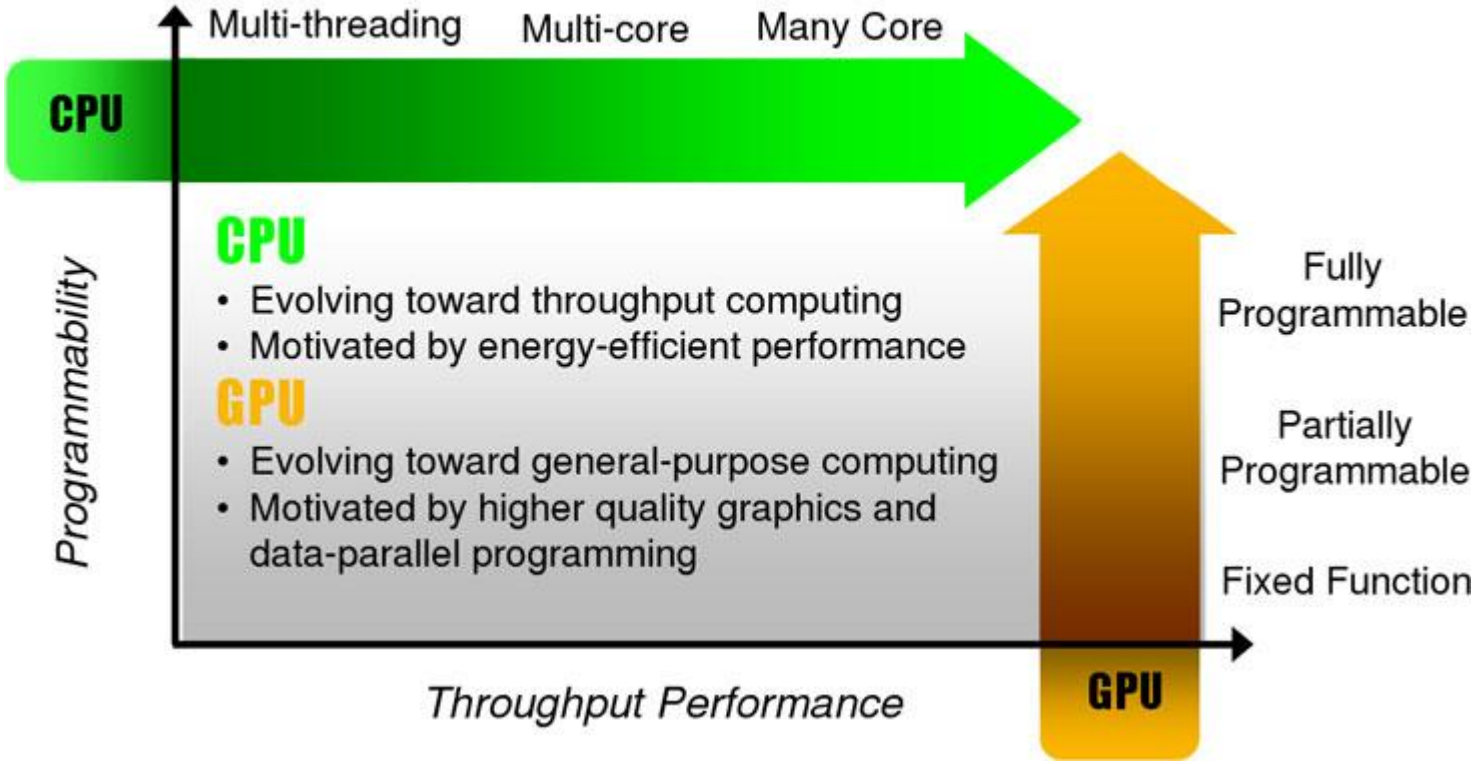
http://jonasboner.com/talks/state_youre_doing_it_wrong/html/all.html

GPU calculation power improves faster



GT200 = GeForce GTX 280	G71 = GeForce 7900 GTX	NV35 = GeForce FX 5950 Ultra
G92 = GeForce 9800 GTX	G70 = GeForce 7800 GTX	NV30 = GeForce FX 5800
G80 = GeForce 8800 GTX	NV40 = GeForce 6800 Ultra	

CPU and GPU convergence



http://images.bit-tech.net/content_images/2009/08/does-nvidia-have-a-future/convergence.jpg

Agenda

- Motivation
- **CUDA Overview**
- Parallelizing the NQueens Problem
- Optimizations
- Evaluation
- Summary / Conclusion

CUDA Programming Model

```
// allocate memory on the graphic card
cudaMalloc((void**)&dataGpu, memSize)

// copy to device memory of the graphic card
cudaMemcpy(pDataGpu, pData, memSize, cudaMemcpyHostToDevice)

// run the calculation kernel
kernel<<<blockSize, threadsPerBlock, sharedMemorySize>>>

// wait for the graphic card threads to finish calculations
cudaThreadSynchronize()

// copy from device memory of the graphic card back to host memory
cudaMemcpy(pData, pDataGpu, memSize, cudaMemcpyDeviceToHost)
```

CUDA Kernel Model

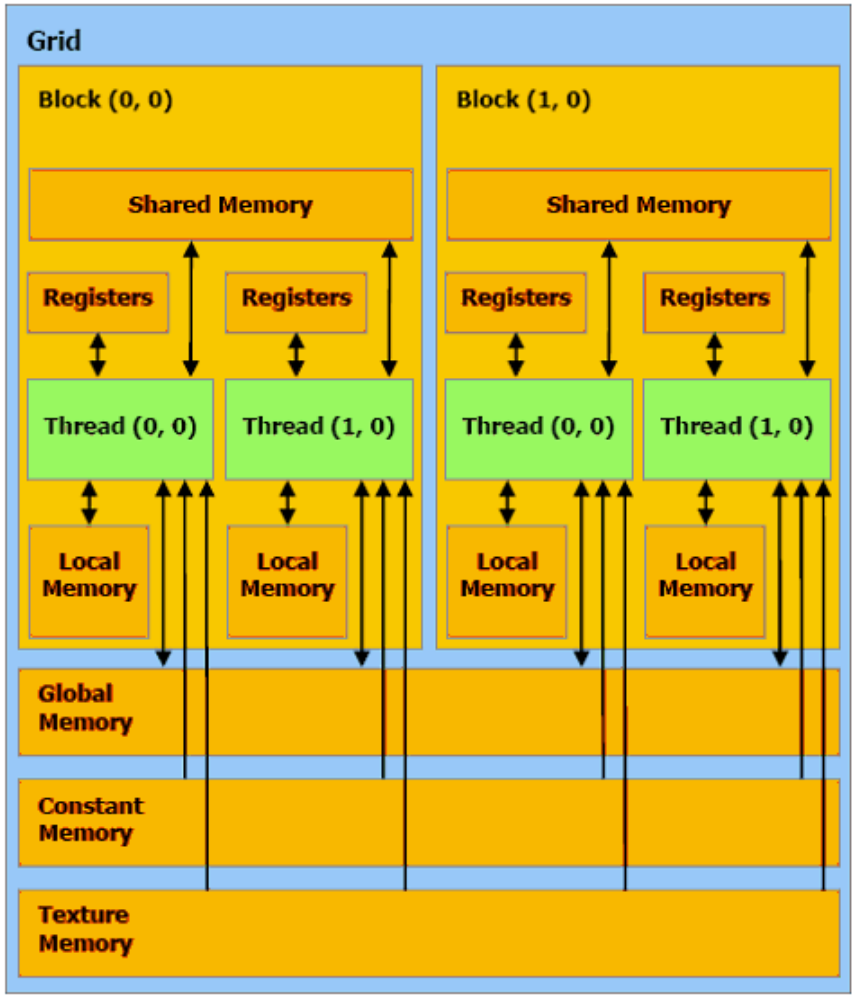
```
// derive absolute thread id from block id and relative thread id
int threadId = blockIdx.x * blockDim.x + threadIdx.x;

// read from array (using thread id as index)
int parameter = dataGpuInput[threadId];

// calculate the result
result = parameter * parameter;

// write to array (using thread id as index)
dataGpuOutput[threadId] = result;
```

CUDA Memory Architecture

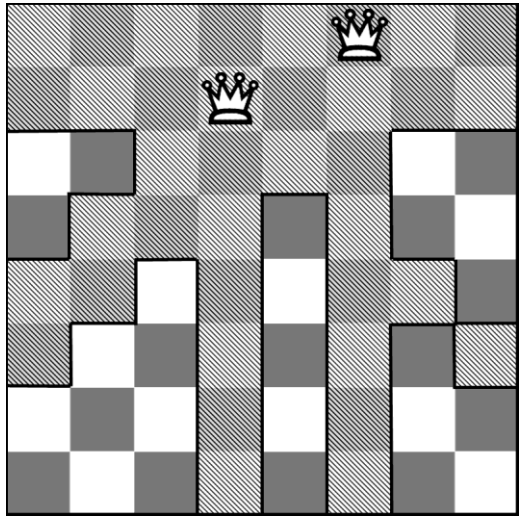


Agenda

- Motivation
- CUDA Overview
- **Parallelizing the NQueens Problem**
- Optimizations
- Evaluation
- Summary / Conclusion

The Whispered World by Daedalic Entertainment / Deep Silver

Precalculated Board Settings



Bitfield row 1 thread 1	Bitfield row 2 thread 1	Bitfield row 3 thread 1	Bitfield row 1 thread 2	Bitfield row 2 thread 2	. . .
. . .	Bitfield row 3 thread 2	Bitfield row 1 thread 3	Bitfield row 2 thread 3	Bitfield row 3 thread 3	. . .
. . .	Bitfield row 1 thread 4	Bitfield row 2 thread 4	Bitfield row 3 thread 4		

Calling the CUDA Kernel

```
int* data = initData(boardsize, depth, &threadCount, &mem_size);
int blockSize = threadCount/threadsPerBlock+1;

cudaMalloc((void**)&data_gpu, mem_size);
cudaMemcpy(data_gpu, data, mem_size, cudaMemcpyHostToDevice);

NqueensCUDA<<<blockSize,threadsPerBlock>>>(boardsize, threadCount, depth,
data_gpu);
cudaThreadSynchronize();

cudaMemcpy(data, data_gpu, mem_size, cudaMemcpyDeviceToHost);

unsigned long long solutionCount = 0;
for(threadId=0;threadId<threadCount;threadId++)
        solutionCount += data[threadId * depth];
```

Our parallelized Nqueens Algorithm (based on J. Somers‘ sequential one)

```
__global__ void NqueensCUDA(int board_size, int threadCount, int depth, int
* data){
        int threadId = blockIdx.x*blockDim.x+threadIdx.x;

        /* mark columns and diagonals that are set */
        int qBitCol[MAX_BOARDSIZE];
        int qBitPosDiag[MAX_BOARDSIZE];
        int qBitNegDiag[MAX_BOARDSIZE];

        /* we use a stack instead of recursion */
        int aStack[MAX_BOARDSIZE + 2];
        register int nStack;

        qBitCol[0] = qBitPosDiag[0] = qBitNegDiag[0] = 0;
        …
```

Initialize Field (Precalculated Setting)

```
for(;numrows < depth;) {
  lsb = data[threadId * depth + numrows];
  bitfield &= ~lsb;

  int n = numrows++;

  /* mark occupied places in next row */
  qBitCol[numrows] = qBitCol[n] | lsb;
  qBitNegDiag[numrows]= (qBitNegDiag[n] | lsb)>>1;
  qBitPosDiag[numrows]= (qBitPosDiag[n] | lsb)<<1;

  aStack[nStack++] = bitfield;
  /* We can't consider positions of queens that already on the board. */
  bitfield= mask &
        ~(qBitCol[numrows]|qBitNegDiag[numrows]|qBitPosDiag[numrows]);
}
```

Agenda

- Motivation
- CUDA Overview
- Parallelizing the NQueens Problem
- **Optimizations**
- Evaluation
- Summary / Conclusion

Optimizing towards shared memory

Optimization Step 1

```
int tx = threadIdx.x * MAX_BOARDSIZE;
  ...
__shared__ int  aStack[(MAX_BOARDSIZE+2)*THREADS_PER_BLOCK];
register int nStack = tx;
```

Optimizations Step 2

```
__shared__ int qBitCol[MAX_BOARDSIZE*THREADS_PER_BLOCK];
__shared__ int qBitPosDiag[MAX_BOARDSIZE*THREADS_PER_BLOCK];
__shared__ int qBitNegDiag[MAX_BOARDSIZE*THREADS_PER_BLOCK];
 ...
qBitCol[tx]=qBitPosDiag[tx]=qBitNegDiag[tx]=0;
```

Using one big shared block

Optimization Step 3

```
extern __shared__ int sharedData[];
...

int iStack = MAX_BOARDSIZE * 4 * tx + 0 * MAX_BOARDSIZE;
int iQBitCol = MAX_BOARDSIZE * 4 * tx  + 1 * MAX_BOARDSIZE;
int iQBitPosDiag = MAX_BOARDSIZE * 4 * tx + 2 * MAX_BOARDSIZE;
int iQBitNegDiag = MAX_BOARDSIZE * 4 * tx + 3 * MAX_BOARDSIZE;

register int nStack = iStack;
...

sharedData[iQBitCol]=sharedData[iQBitPosDiag]=sharedData[iQBitNegDiag] = 0;
```

Reducing necessary block size

Optimization Step 4

```
int abs = board_size - depth;
int iQBitCol = abs * 4 * tx + 0 * abs;
int iQBitPosDiag = abs * 4 * tx + 1 * abs;
int iQBitNegDiag = abs * 4 * tx + 2 * abs;
int iStack = abs * 4 * tx + 3 * abs;
```

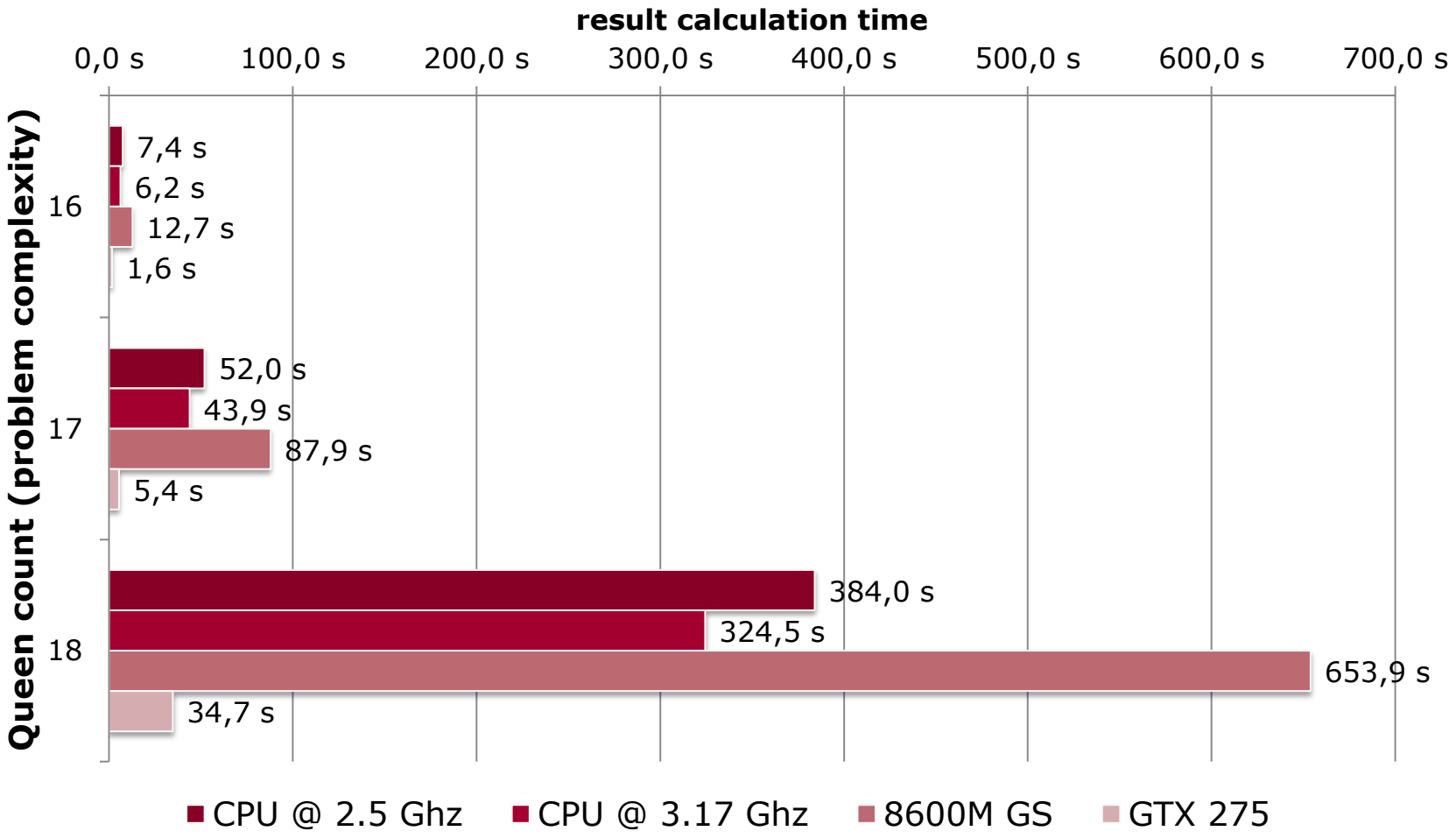
Agenda

- Motivation
- CUDA Overview
- Parallelizing the NQueens Problem
- Optimizations
- **Evaluation**
- Summary / Conclusion

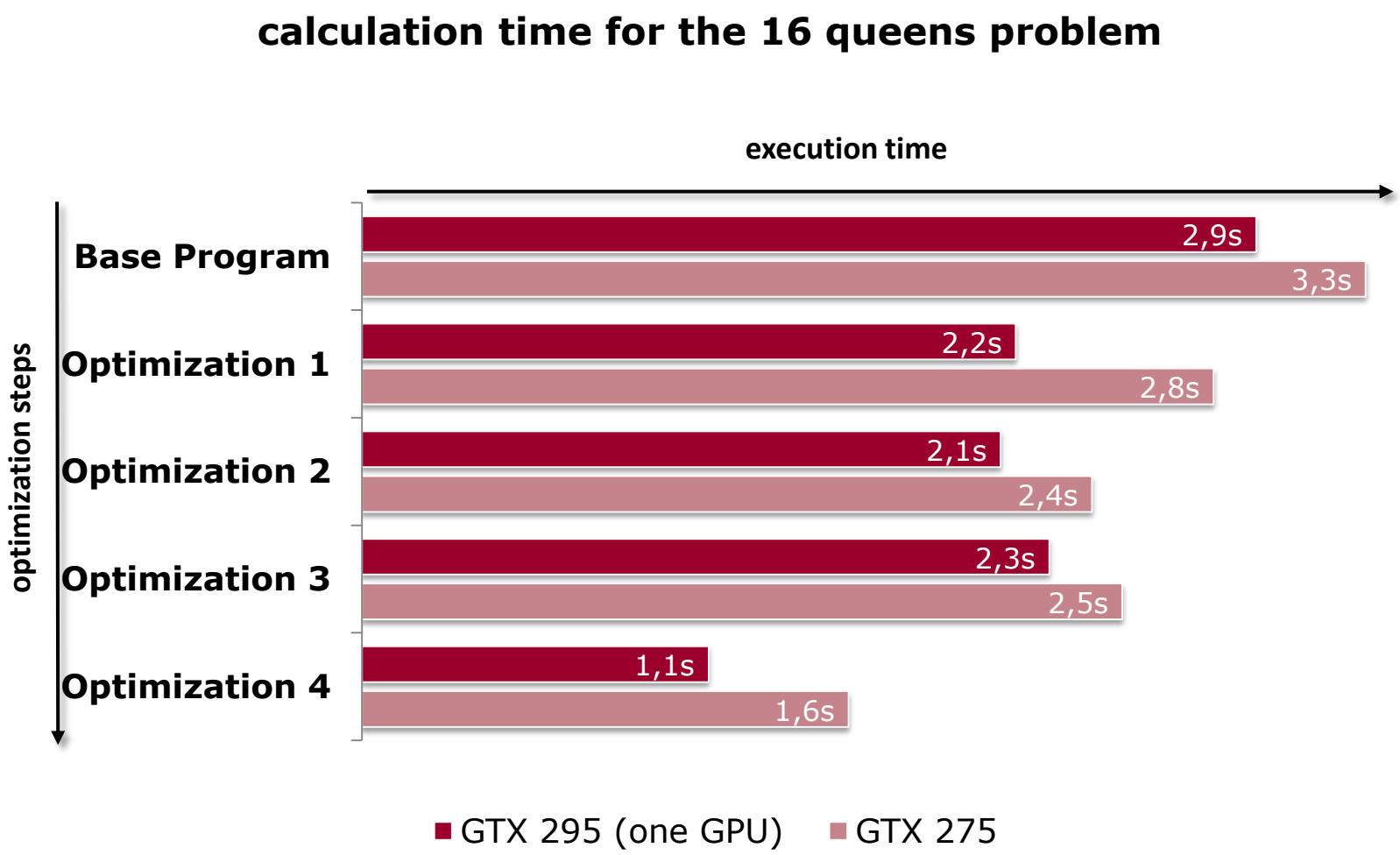
Testsystems

Component	Description
test system 1	
CPU	Intel Core 2 Duo E8500, 2x3.17 GHz
Chipset	Intel P45
Graphics card	Gainward GeForce GTX 275
RAM	4GB (DDR2-1000)
test system 2	
CPU	Intel Core 2 Duo Mobile T9300 2x2.5 GHz
Chipset	Intel PM965
Graphics card	Nvidia GeForce 8600M GS
RAM	3 GB (DDR2-800)
test system 3.1	
CPU	Two Intel Xeon E5520 4x2.26 GHz
Chipset	Intel 5520
Graphics card	Nvidia GeForce GTX 295
RAM	6 GB (DDR3-1066)
test system 3.2	
CPU	Two Intel Xeon E5520 4x2.26 GHz
Chipset	Intel 5520
Graphics card	Nvidia Quadro NVS 295
RAM	6 GB (DDR3-1066)

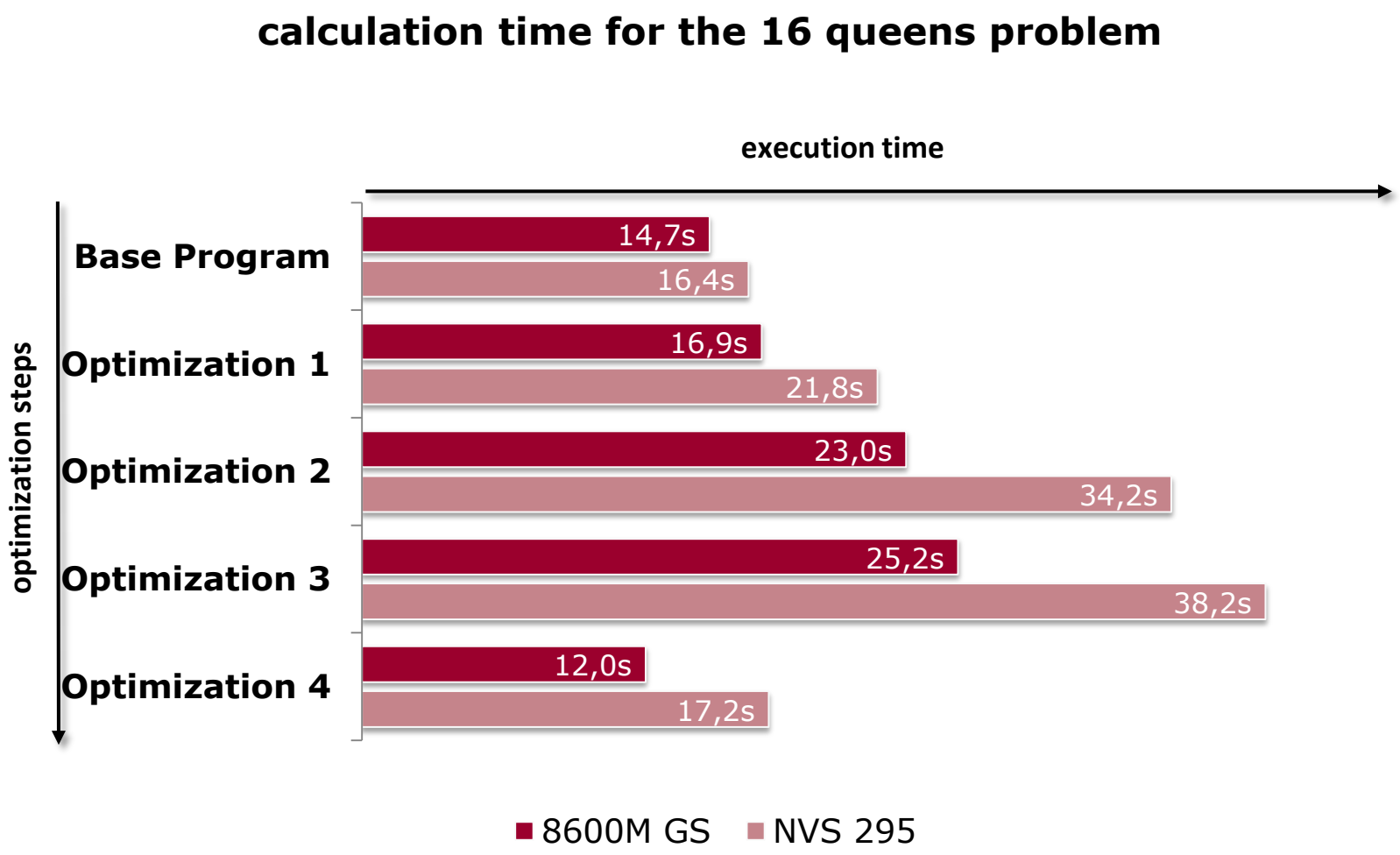
Performance



Optimization Impact for CUDA Capability 1.3



Optimization Impact for CUDA Capability 1.1



Agenda

- Motivation
- CUDA Overview
- Parallelizing the NQueens Problem
- Optimizations
- Evaluation
- **Summary / Conclusion**

Summary / Conclusion

- Study GPUs now to know some characteristics of the CPUs of the future
- Nowadays GPUs are very fast and easy to program …
  … but the resulting performance is very hard to predict due to different hardware capabilities of different graphic card series.
  - Deep understanding of the target platform
  - Its not clear from looking at the code whether accessing a variable will be fast or slow
- Lots of additional restrictions:
  - Long running functions -> driver will crash
  - reusing of kernel-code in normal program is impossible
  - No recursion