Neighborhood Evaluation on GPU for the DCVRP

Discrete optimization needs heterogeneous computing

Christian Schulz, Geir Hasle
Department of Applied Mathematics, SINTEF ICT, Oslo, Norway

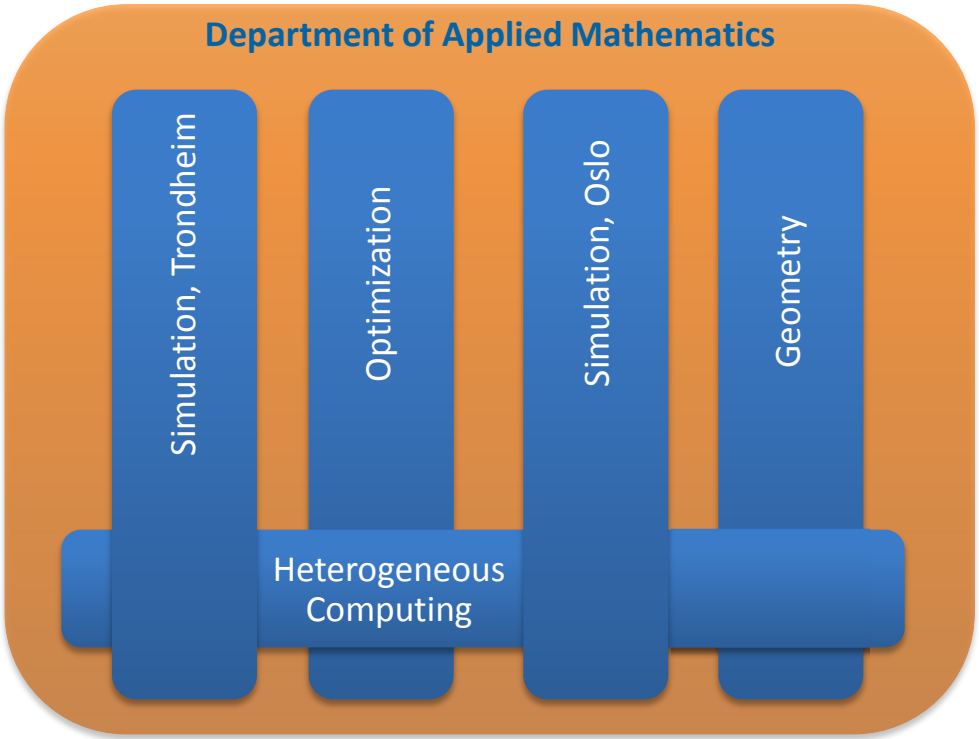
ROUTE2011

Sitges, Spain, June 1, 2011

Outline

- Performance in Discrete Optimization
- Hardware developments, and prospects
- Accelerators and heterogeneous computing
- A GPU based VRP solver
- Incremental improvement of implementation
- Extension to truly heterogeneous computing
- Conclusions

Department of Applied Mathematics



- Offices in Oslo and Trondheim
- Consists of 5 research groups
  - Geometry
  - Optimization
  - Simulation
  - Visualization
  - Heterogeneous computing
- Key figures 2009
  - 38 employees
  - 45 MNOK turnover

Performance in Discrete Optimization

- DOPs computationally hard
- Tremendous increase in DOP solving ability
- Illustration: Commercial LP solvers*
- Speedup factor roughly 1.000.000 1987-2000
- Factor 1000 better methods
- Factor 1000 faster computers

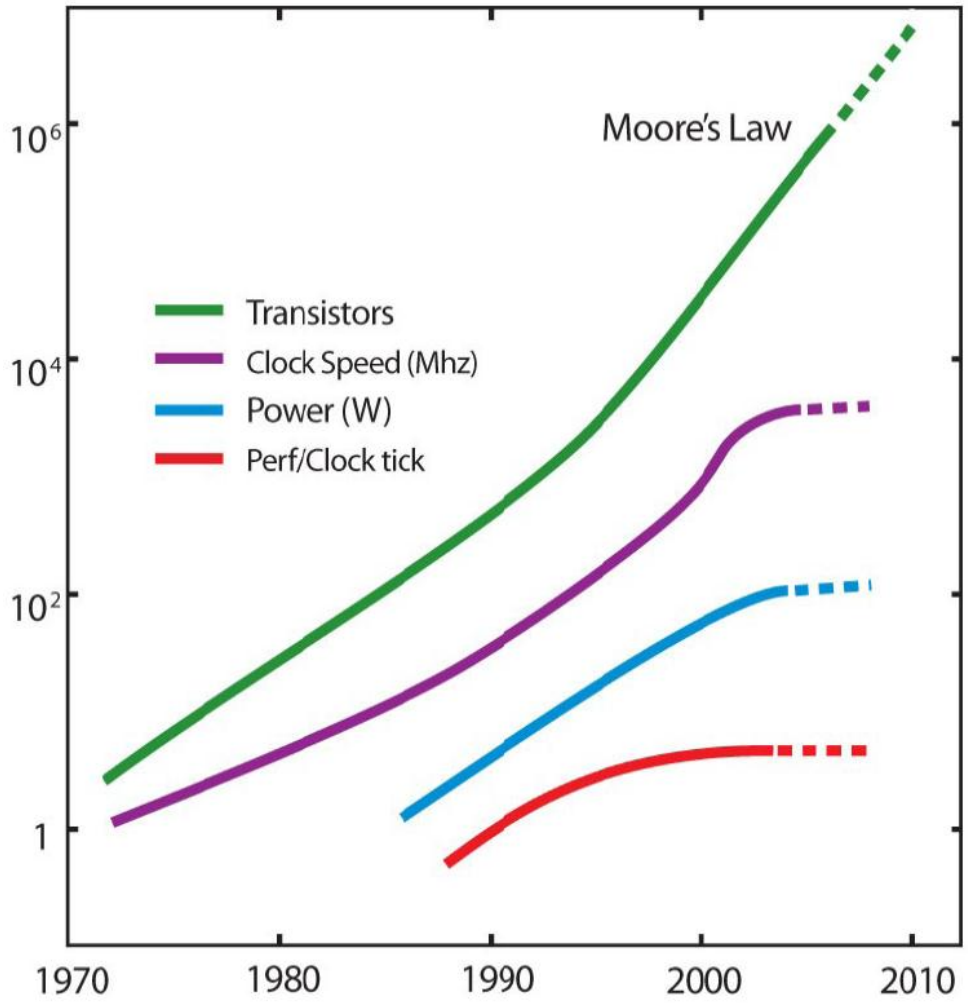
- There is still a performance bottleneck in industry

*Bixby R.E. (2002). Solving Real-World Linear Programs: A Decade and More of Progress. Oper. Res. 50(1), pp. 3-15.

The Beach Law [Gottbrath et al. 1999]

One way of doubling the performance of your computer program is to go to the beach for 2 years and then buy a new computer.

Processor development 1970-2010



"The number of transistors on an integrated circuit for minimum component cost doubles every 24 months"

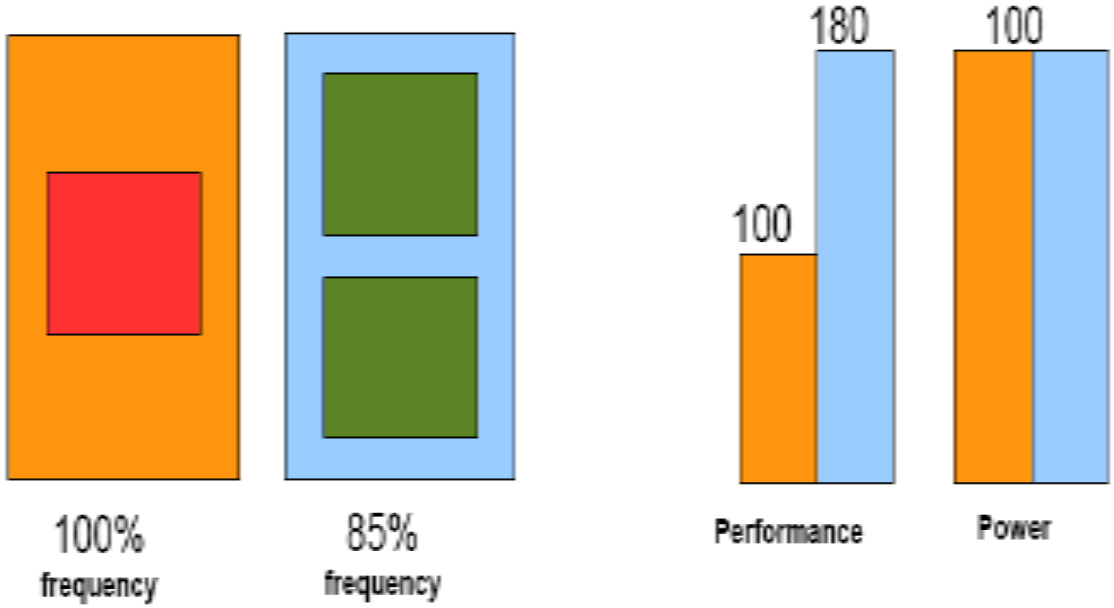
– Gordon Moore, 1965.

What happened?

- Moore’s law at work, expected to hold until 2030 ...
- The Beach Law was valid until about 2005 ...
- Heat dissipation etc. stopped it
- PC computing power still benefits from Moore’s law
- Multi-core processors for task parallelization (multi-threading, shared memory)
- Accelerators for data parallelization (stream processing)

- Drastic change in the development of processors

Multi-core processors

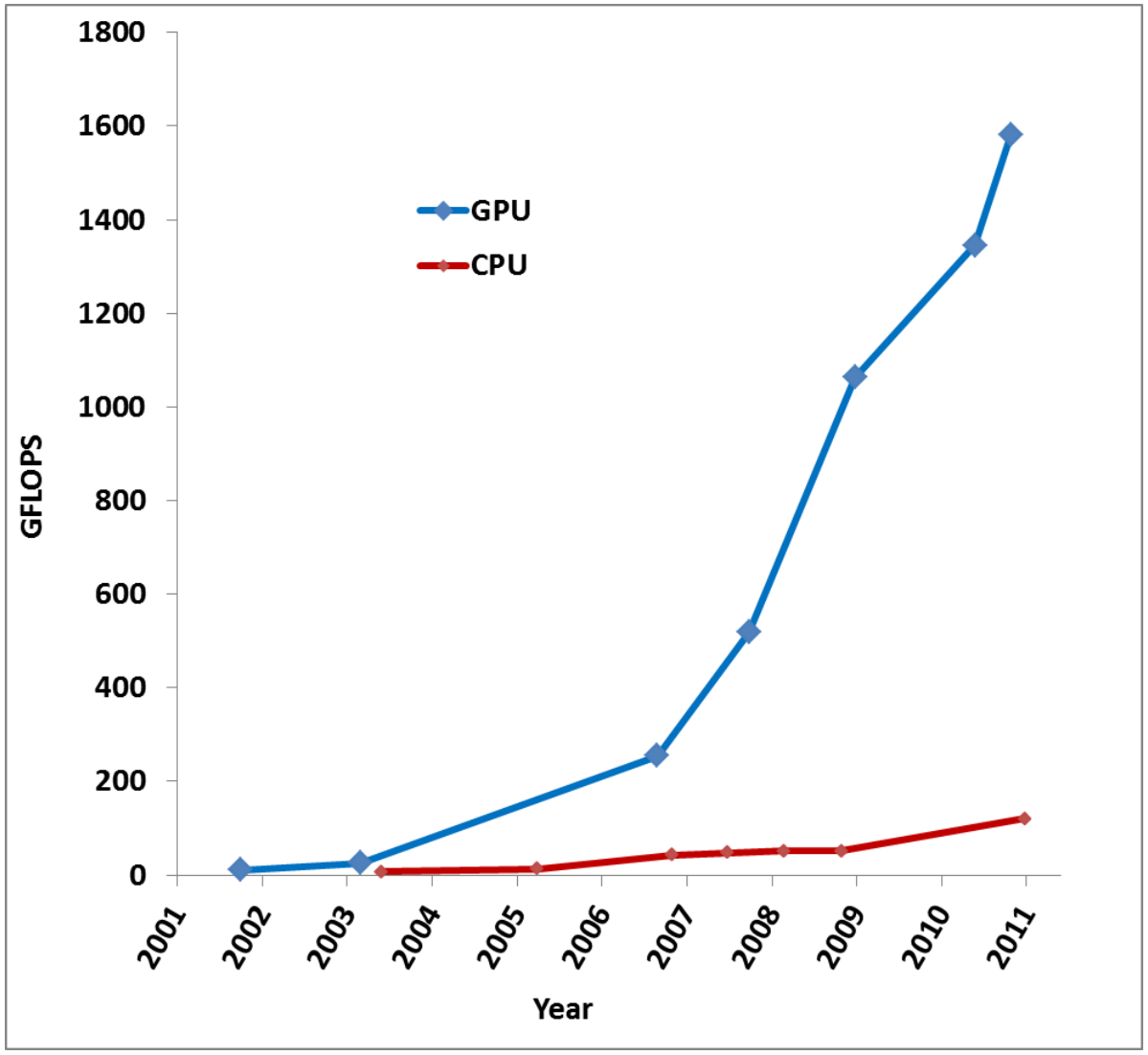


- Heat dissipation varies with clock frequency cubed
- 2 cores, reduced frequency, same heat dissipation
- 70% higher computing performance **if you can exploit it**
- **Sequential programs will run slower**

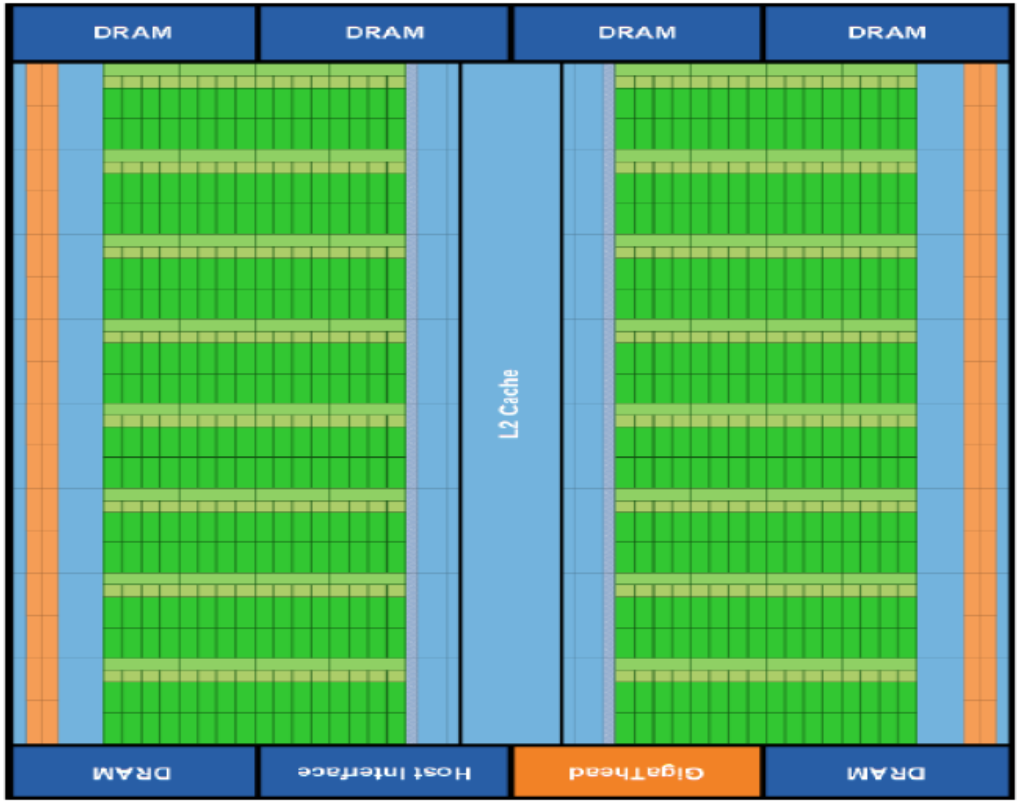
Stream processing accelerators

- The graphics card was the origin
- Developent driven by gaming industry
- Computing power increases rapidly
- Programmability improves rapidly
- Libraries, debugging, performance, profiling tools
- Single Program Multiple Data
- Massively parallel, thousands of threads
- You need to
  - understand the architecture
  - worry about code diversion
  - worry about memory latency
  - worry about …

GPU vs CPU performance

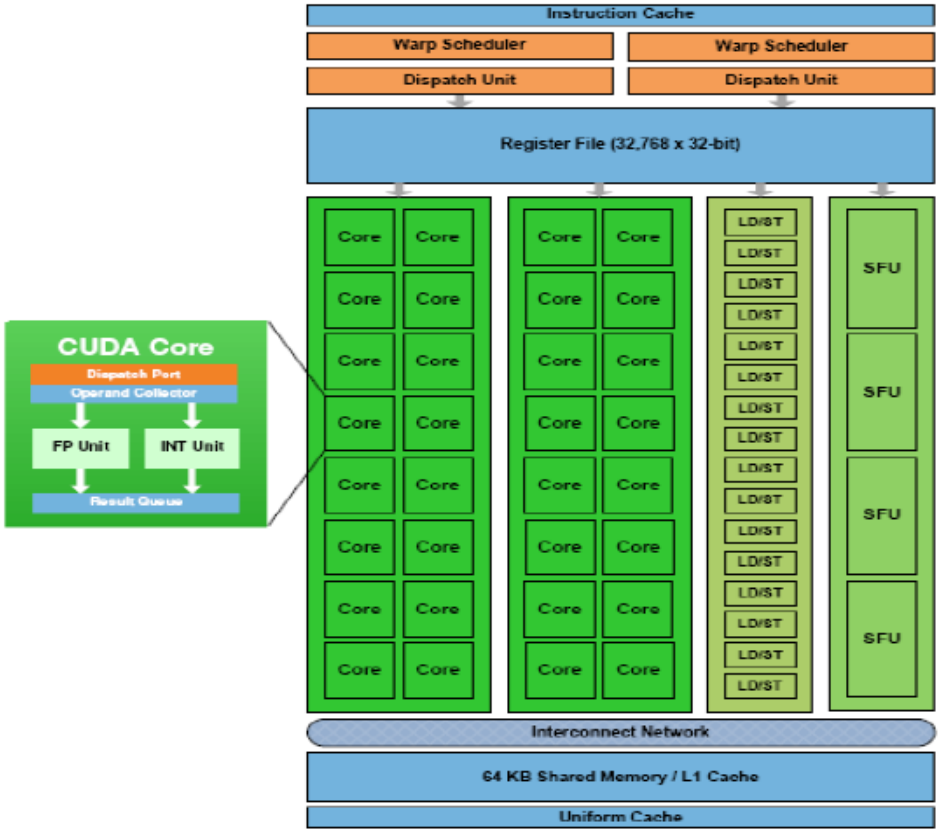


The GPU – NVIDIA Fermi Architecture



16 streaming multiprocessors are positioned around a common L2 cache

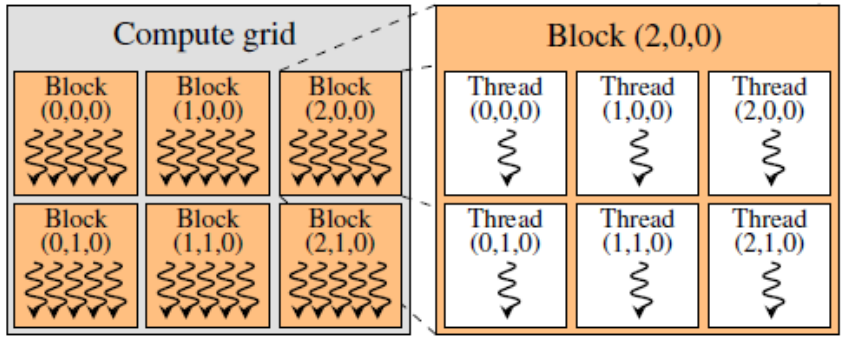
The GPU – NVIDIA Fermi Architecture

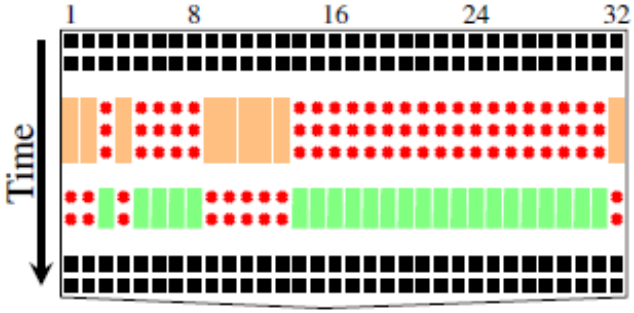


Each of the16 Streaming Multiprocessors (SMs) has 32 cores, 512 cores in total. Each core run the same program («kernel»), with individual data and individual code flow (SPMD). Divergence means serialization. Need more threads than cores to hide latency, typically >512 threads for each SM, say 10.000. One may run multiple kernels concurrently.

Kernel execution

- Kernel
  - Executed on a compute grid
    - Consisting of blocks
      - Each with a number of threads
- Max # threads/block: 1024
- All threads in a block on same SM
- Different blocks may execute on different SMs
- Block threads split in warps of 32 threads
- Warp serialization and masking, minimize code divergence






```
//Non-divergent code
if( c > 0 ) {
    x = e / c;
    x *= s;
    z = x - a;
} else {
    x = 0;
    z = a;
}
//Non-divergent code
```

Programming GPUs

- OpenCL
  - API for multi-platform shared memory multiprocessing
  - C, C++, Fortran
  - Open standard, Khronos group
- CUDA
  - C++-like language
  - proprietary (NVIDIA)
  - libraries
  - development tools (debugger, profiler, …)

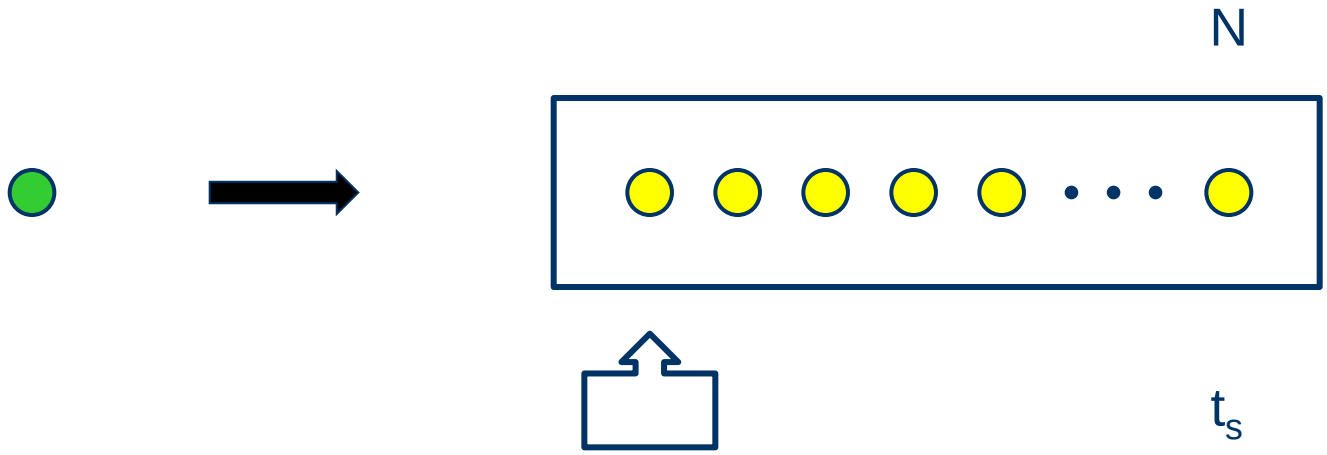
Exploiting the GPU

- Games
- Matrix and vector operations
- Scientific simulation and visualization http://www.youtube.com/babrodtk
- http://www.nvidia.co.uk/object/cuda_apps_flash_new_uk.html#

- Local search
- Genetic algorithms

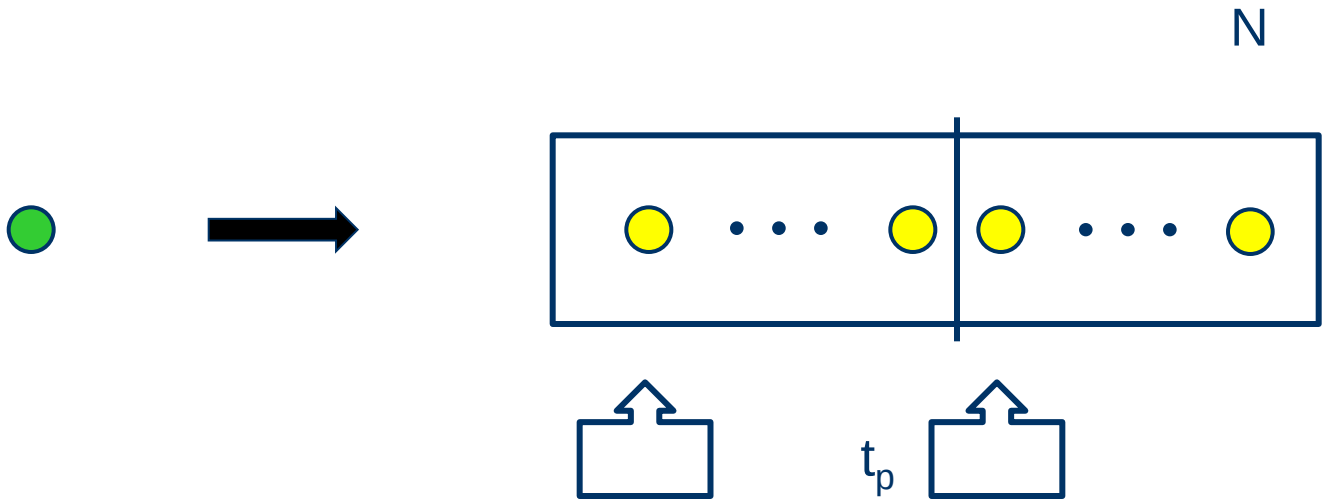
- Simple idea: evaluation of neighbors / individuals

Local Search
- Sequential evaluation of neighborhood



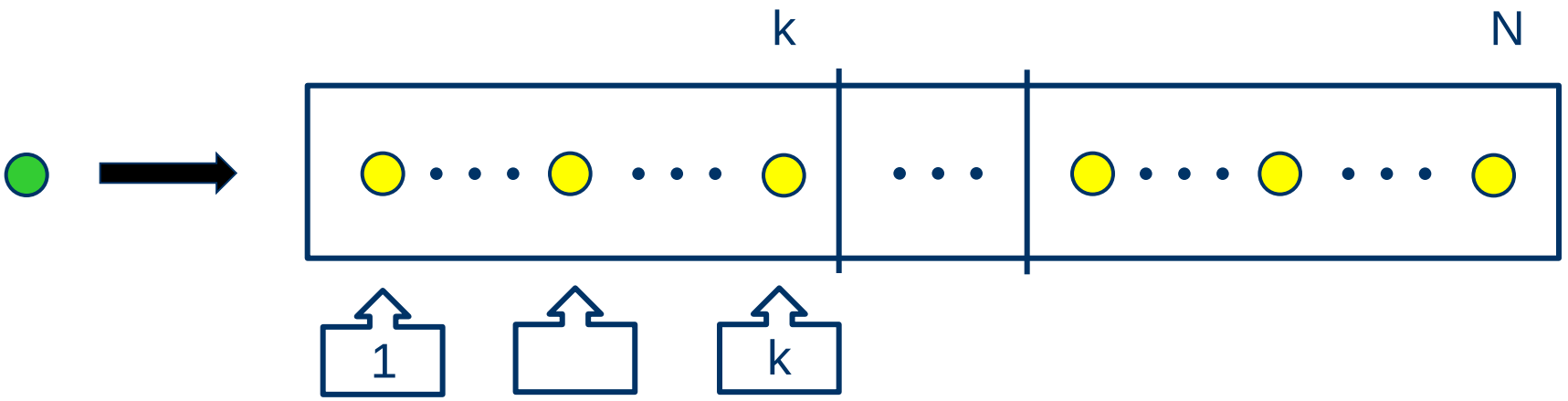
Time for one iteration: $t_s N$

Local Search
– Task parallel evaluation of neighborhood 2 cores



Time for one iteration: $t_pN/2$

Local Search
– Data parallel evaluation of neighborhood



simultaneous threads: k

Time per evaluation: t_g

Time for one iteration: $t_g N/k$

Heterogeneous computing

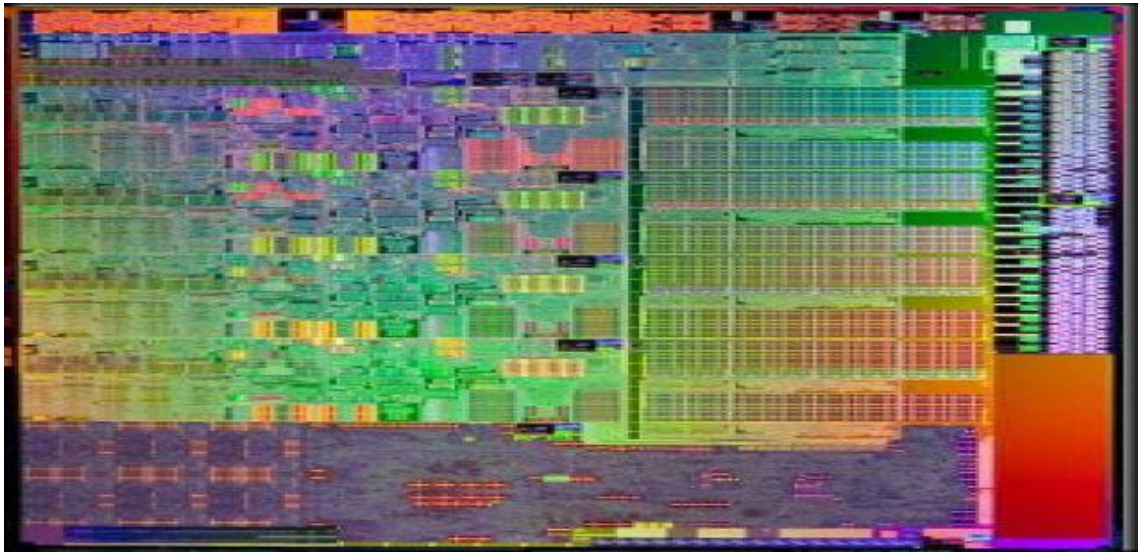
- **Heterogeneous computing systems**: electronic systems that use a variety of different types of computational units.
- Current and future PCs are parallel and heterogeneous
- **Heterogeneous computing** aims to combine the parallelism of traditional multi-core CPUs and accelerators to deliver unprecedented levels of performance

"GPUs have evolved to the point where many real-world applications are easily implemented on them and run significantly faster than on multi-core systems. Future computing architectures will be hybrid systems with parallel-core GPUs working in tandem with multi-core CPUs."

Prof. Jack Dongarra, Director of the Innovative Computing Laboratory
The University of Tennessee

Supercomputer on a chip
Single die heterogeneous processors

- AMD Fusion
- Intel Sandy Bridge

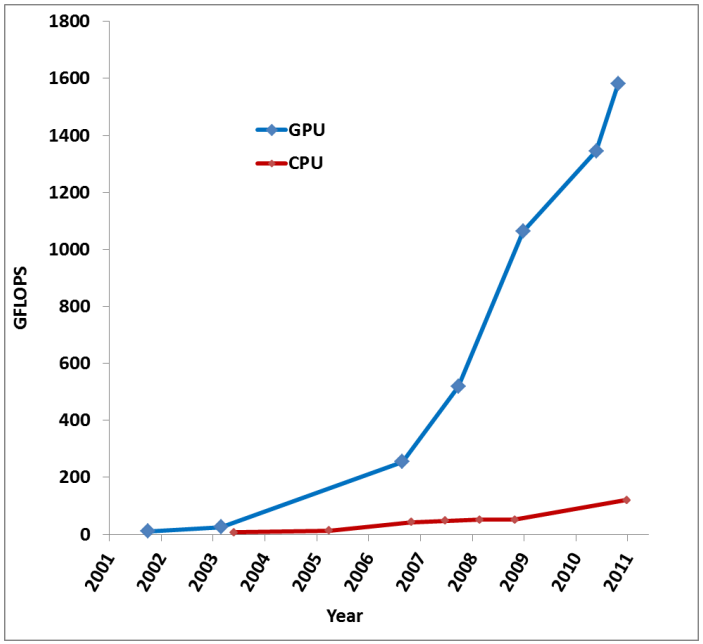


Why bother?

- Exploit present hardware
- Profit from the future increase of processor power

- Robustness
- Larger-size, richer, more integrated problems
- Stochastic models
- Multi-criteria problems
- Real-time applications
- New ideas in optimization

- Automated parallelization?
- Tool vendors?



Literature

- Van Luong T.V., Melab N., Talbi E.-G.: **Neighborhood Structures for GPU-based Local Search Algorithms.** Parallel Processing Letters, Vol. 20, No. 4, pp. 307-324, December 2010
- Hasle G., Kloster O., Riise A., Schulz C., Smedsrud M.: **Using Heterogeneous Computing for Solving Vehicle Routing Problems**. Extended abstracts TRISTAN VII, Tromsø, Norway, June 20-25, 2010.
- Schulz C., Hasle G., Kloster O., Riise A., Smedsrud M.: **Parallel local search for the CVRP on the GPU.** META'10, Djerba, Tunisia, October 28 2010
- Special session «**Metaheuristics on graphics hardware**» at META'2010 http://www2.lifl.fr/META10/pmwiki.php?n=Main.InfoMGH
- JPDC Special Issue: Metaheuristics on GPU, Expected April 2012

Activities at SINTEF Applied Math.

- PDA-based simulation, geometry, visualization since 2003
- NVIDIA CUDA Research Center
- Collab project 2009-2012
- Task parallelization of industrial VRP Solver «Spider»
- Experimental VRP solver: «Camel Spider»
- Project workshops
- META'2010 special session «Metaheuristics on graphics h
- JPDC special issue «Metaheuristics on GPU»

Earlier work – metaheuristics on GPU

- Basic implementations
- Performance not so impressive
- Speedup comparison with naive CPU implementation

Goals

- Long term
  - VRP solver based on heterogeneous computing
  - Modern PC, multi-core CPU + stream processing accelerator
  - Self-adaptability
- Step1
  - How efficient can we make local search using the GPU?
  - Goal is speed, not solution quality

Experimental setup – LS for DCVRP

- Giant Tour Representation
- Resource Extension Functions (REFs)
- Segment hierarchy for constant time neighbor evaluation
- 2-opt and 3-opt on the full giant tour representation
- 10 standard instances from the literature, 57-2401 nodes
- NVIDIA GTX480 (Fermi architecture)
- CUDA v3.2

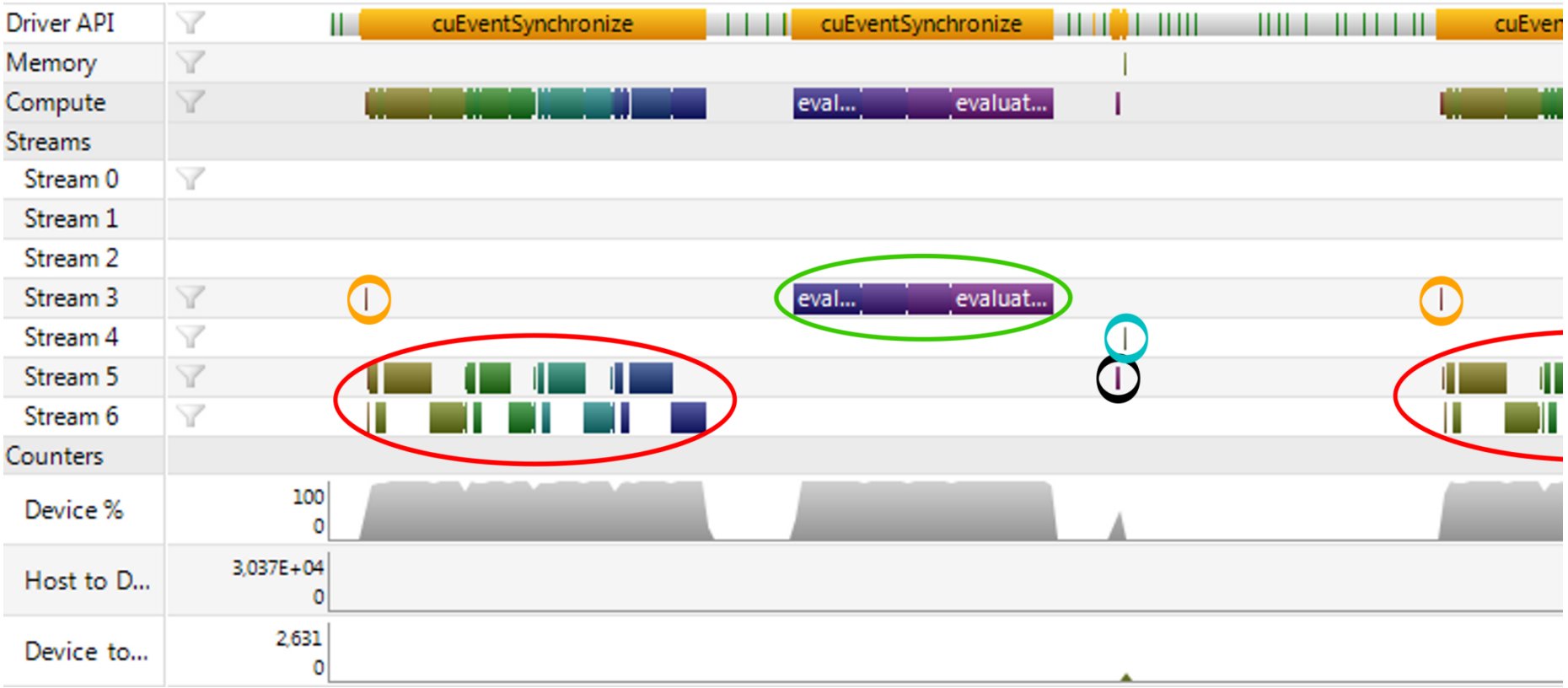
Iterative improvement process

- Basic, Benchmark Version
- Iterate
  - experiments
  - speedup over incumbent version
  - analysis of performance
  - identify problems, focus on some implementation aspect
  - explore alternative remedies
- Until stop criterion …

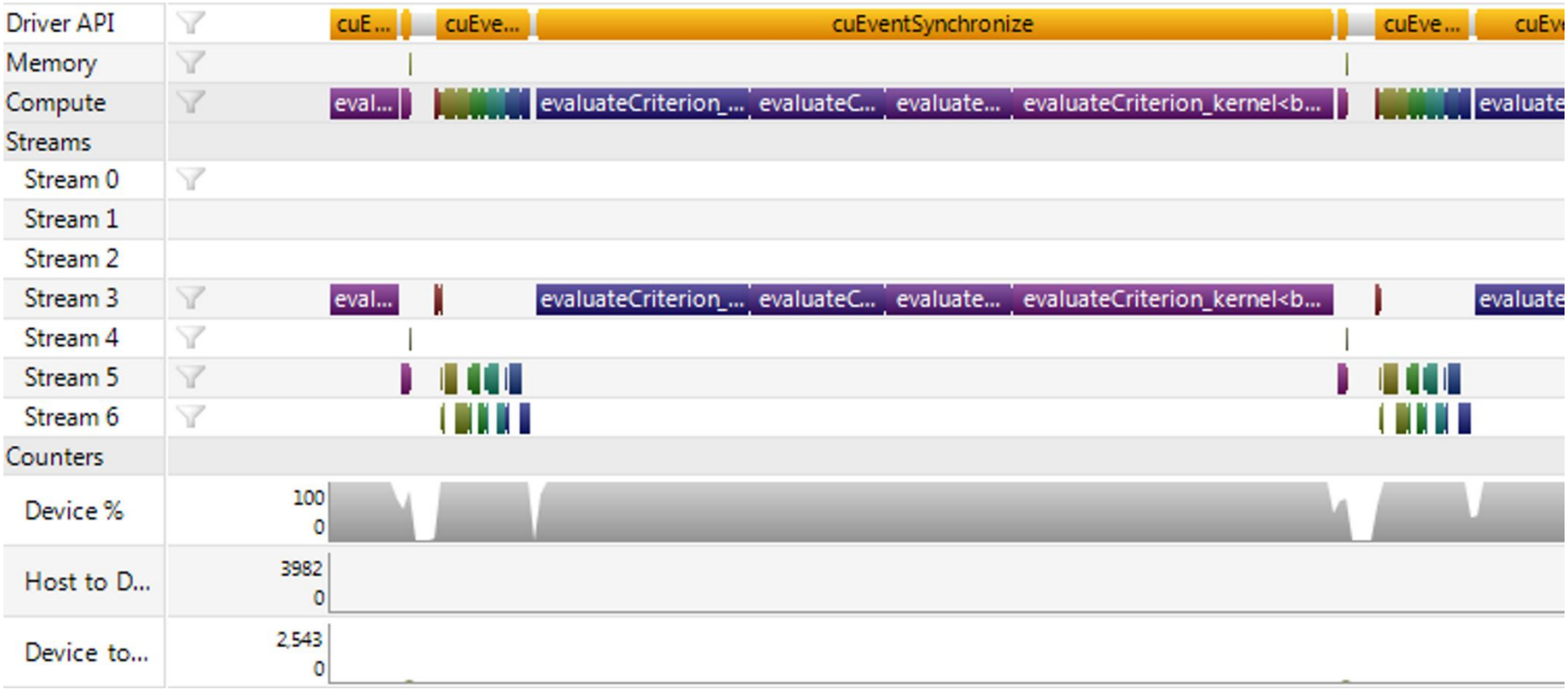
The algorithm

- Setup problem instance data on CPU
- Copy problem instance data to GPU
- Create initial solution on CPU
- Copy initial solution to GPU
- Evaluate initial solution on CPU
- Create k-opt mapping on CPU
- Copy k-opt mapping to GPU
- do
  - Create segment hierarchies on GPU
  - Evaluate all constraints and objectives on GPU
  - Find best neighbor on GPU
  - Execute best move on GPU
  - Copy best move to CPU
  - Execute best move on CPU
  - Evaluate new current solution on CPU
- until local optimum or stop criterion

2-opt iteration, 400 nodes, benchmark



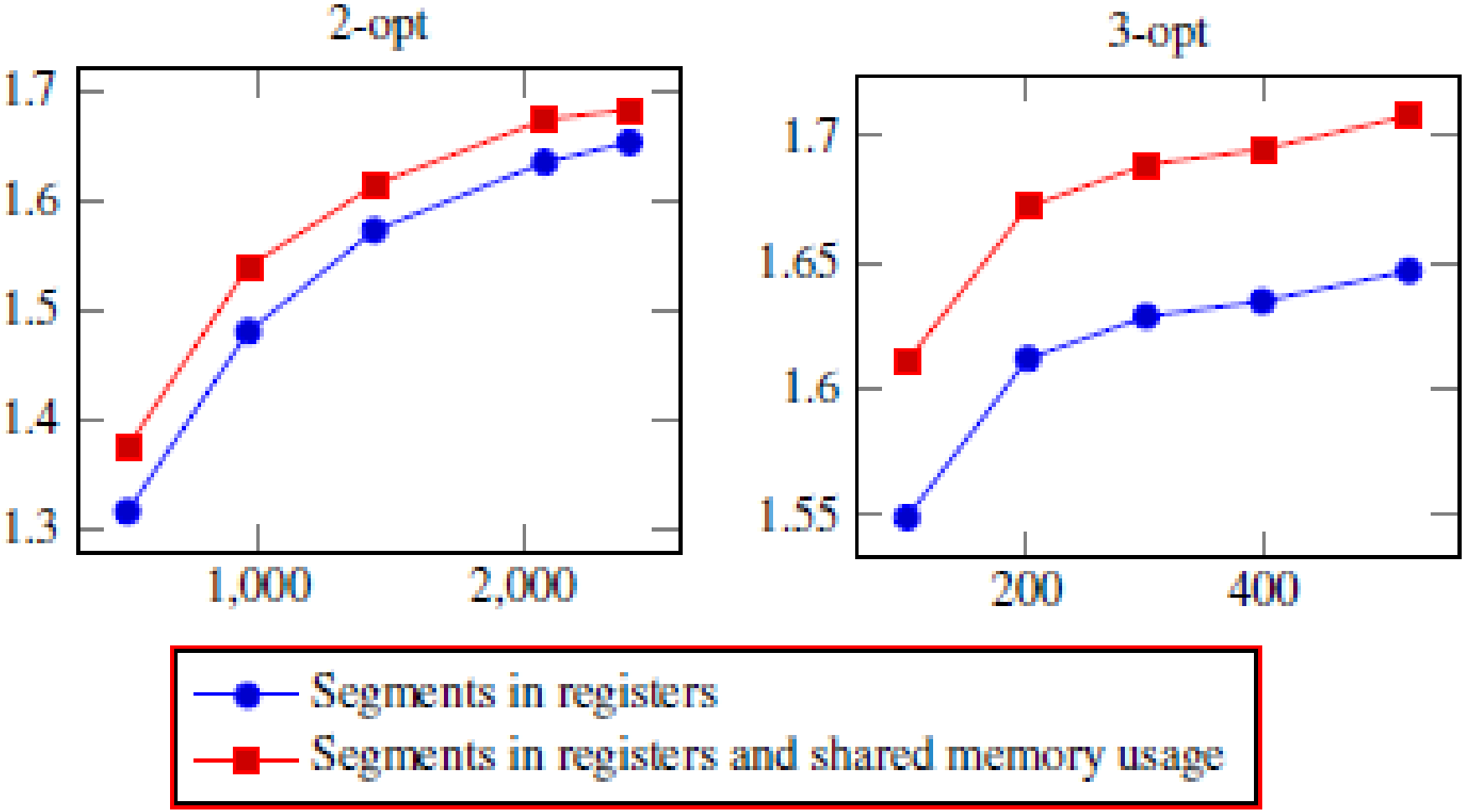
2-opt iteration, 2400 nodes, benchmark



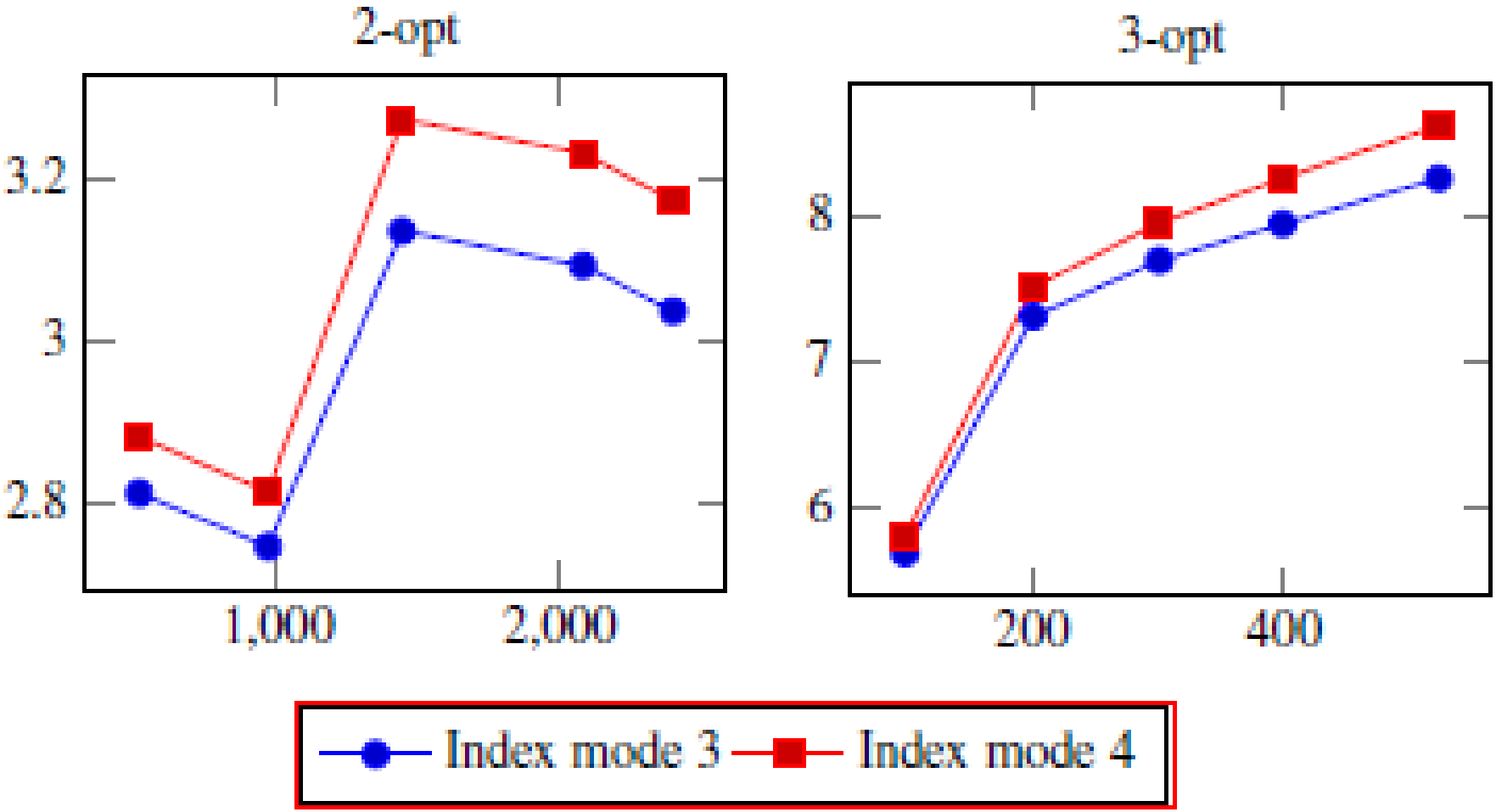
Improvements

- Segments in registers
- Shared memory
- Avoid expensive instructions
- Block size
- Datastructures
- Indexing of segments
- Thread index vs neighbor mapping
- Depth of segment hierarchy
- Combined evaluation
- More clever synchronization of CPU and GPU

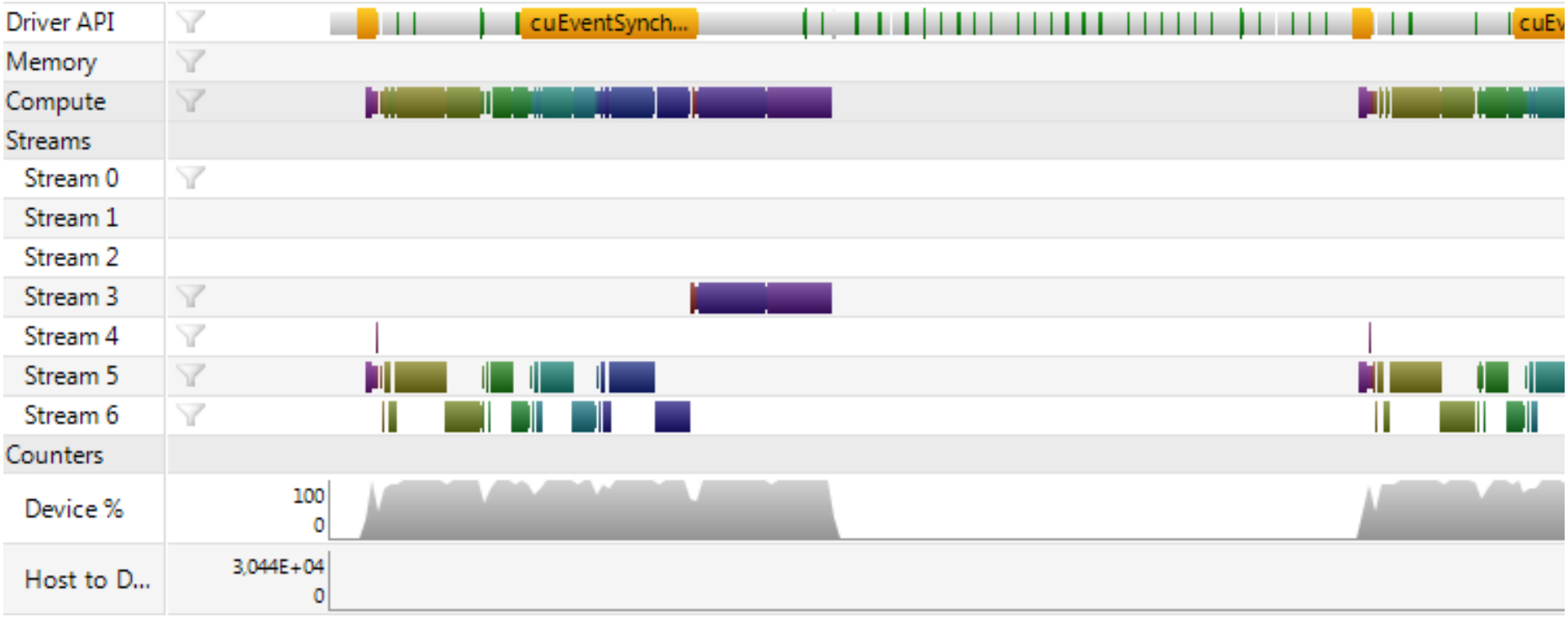
Memory management



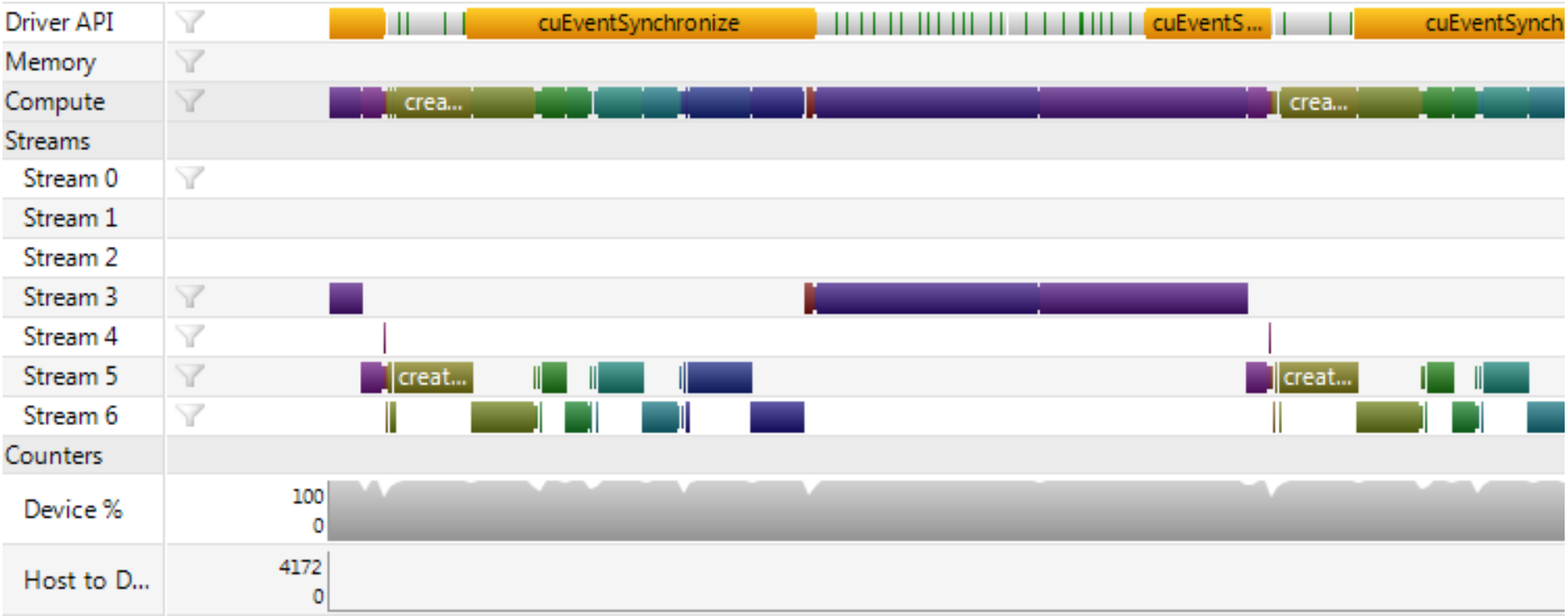
Overall speedup



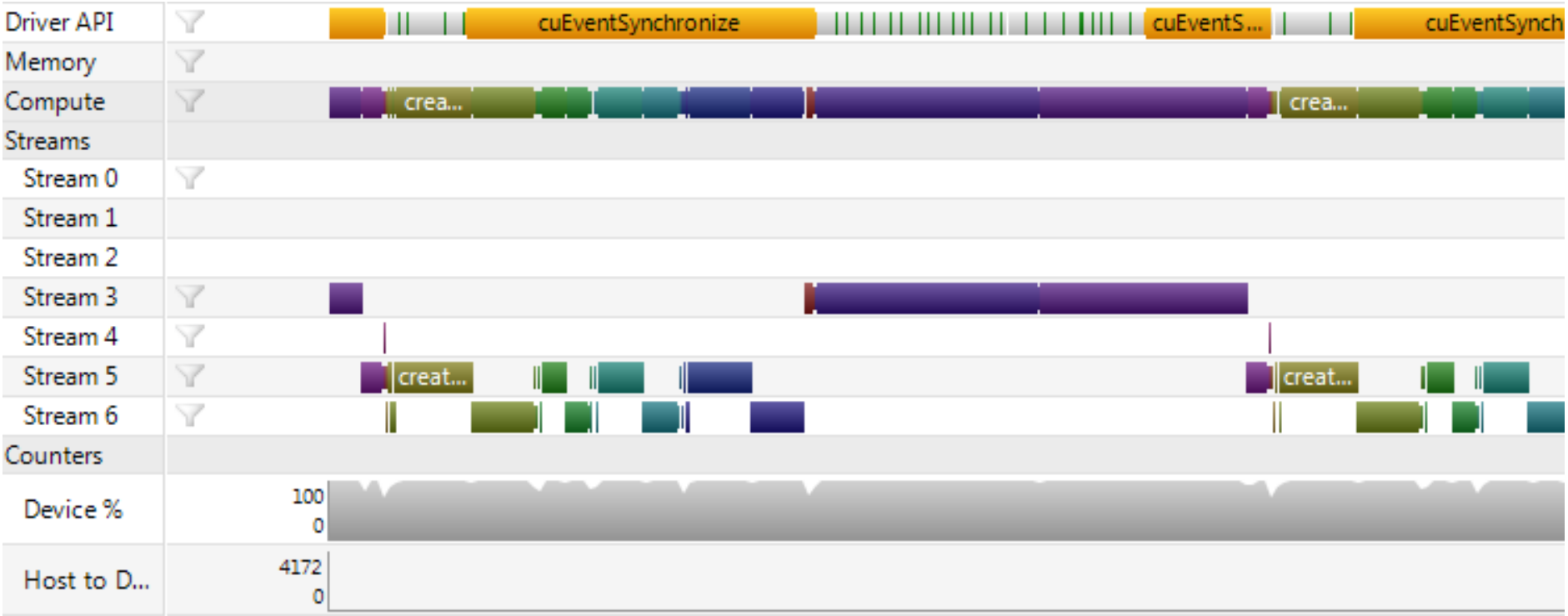
2-opt iteration 400 nodes, final version



2-opt iteration 1000 nodes, final version



3-opt iteration 735 nodes, final version



Insights

- Efficient kernel is important
- Synchronization CPU/GPU is important
- Keep the GPU busy
- Neighborhood size should be large enough
  - 2-opt: 900 nodes
  - 3-opt: 110 nodes
- Up to an order of magnitude speedup gained by careful tuning

Results

- 9 billion 3-opt moves generated and evaluated in 36 s (4 ns per move, 8 clock cycles in a 2 GHz CPU core).
- Speedup factor vs. serial CPU up to almost 1000

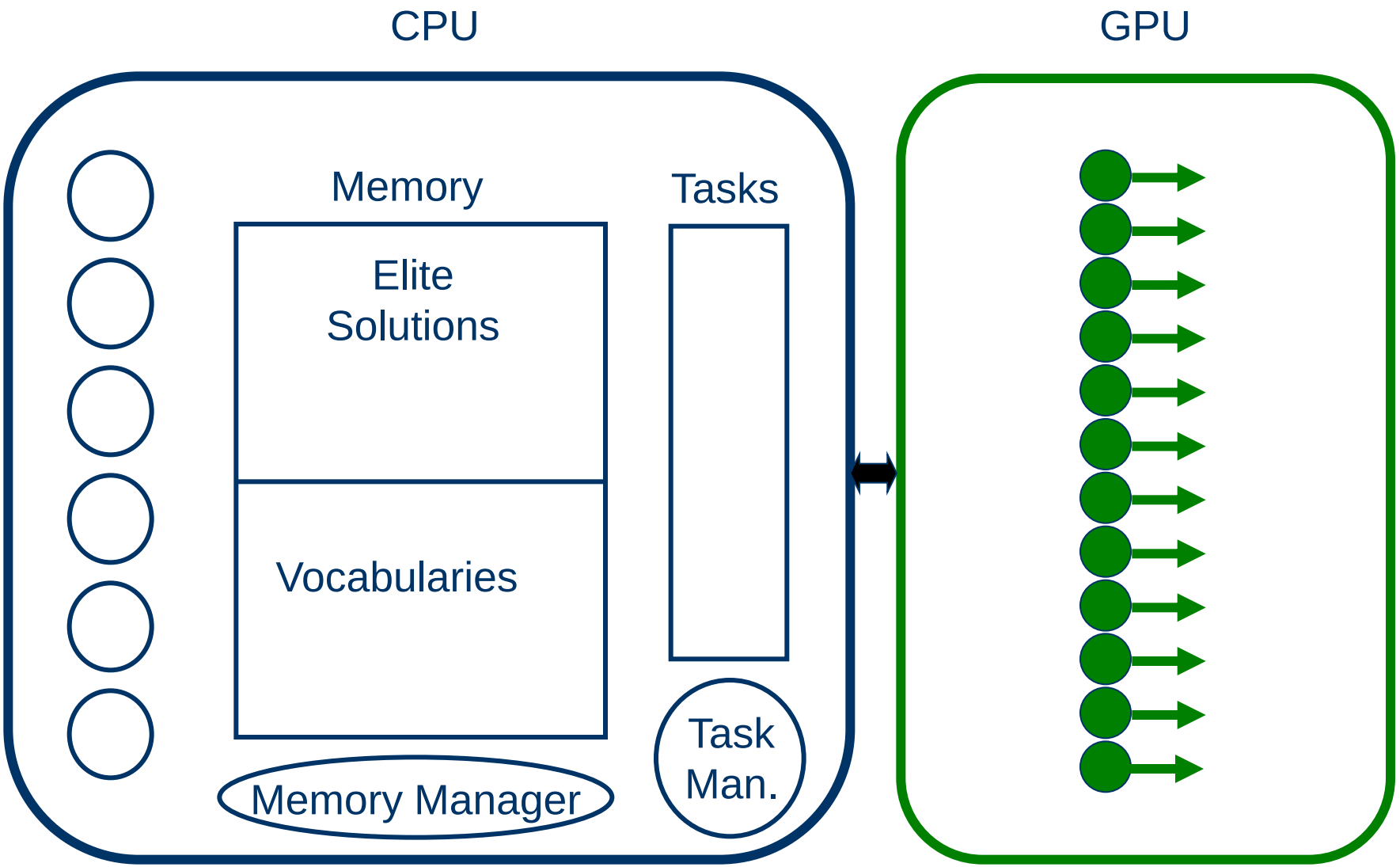
- The GPU is a powerful intensification machine

- The CPU is almost idle …

Ideas – Heterogeneous DOP Computing

- Goal: Balanced use of available computing devices
- Self-adaptation to available hardware
- The GPU is a mean intensification machine
  - Local Search
  - Large Neighborhood Search
  - Variable Neighborhood Search
  - ...
- CPU used for more «sophisticated» tasks

Sketch of labor division – VRP Solver



Neighborhood Evaluation on GPU for the DCVRP

Discrete optimization needs heterogeneous computing

Christian Schulz, Geir Hasle
Department of Applied Mathematics, SINTEF ICT, Oslo, Norway

ROUTE2011

Sitges, Spain, June 1, 2011