GPU based Local Search for the DCVRP

Christian Schulz, Geir Hasle, Oddvar Kloster, Atle Riise and Morten Smedsrud

SINTEF ICT

12 May 2011

Outline

Motivation

Vehicle Routing Problem

- Still gap between requirements and performance
- Variants of large neighborhood search, variable neighborhood search, iterated local search proven effective

Why parallelize local search

- Local search is an essential part of more advanced strategies such as metaheuristics
- Embarrassingly parallel: Moves independent from each other

$\Rightarrow$ Potential for significant speed up

Why GPU

- High computational power and memory bandwidth
- Cheap

Model

(D)CVRP

- Given: depot & customer nodes, travelling costs, vehicle capacity, customer demands (, number of vehicles, maximal length per route)
- Wanted: Feasible route(s) with minimal length

Model

- Based on paper "A Unified Modeling and Solution Framework for Vehicle Routing and Local Search-based Metaheuristics" by Stefan Irnich, INFORMS JOURNAL ON COMPUTING, Vol. 20, No. 2, Spring 2008, pp. 270-287
- Solution represented as a giant tour
- Use of classical resource extension functions to model capacity constraint and maximal length per route constraint
  $\Rightarrow$ Constant time move evaluation

Classical Resource extension function

- Resource vector $\mathbf{T} \in \mathbb{R}^n$
- Each node has a associated resource interval $[\mathbf{a}_i, \mathbf{b}_i]$
- A classical REF models change in resource from i to j:
  $$\mathbf{f}_{ij}(\mathbf{T}) = \mathbf{T} + \mathbf{t}_{ij} \quad \text{or} \quad \mathbf{f}_{ij}(\mathbf{T}) = \max(\mathbf{a}_j, \mathbf{T} + \mathbf{t}_{ij})$$
- A path is feasible if for each node i there exists a resource vector $\mathbf{T}_i \in [\mathbf{a}_i, \mathbf{b}_i]$ s.th.
  $$\mathbf{f}_{i,i+1}(\mathbf{T}_i) \leq \mathbf{T}_{i+1}$$

Classical Resource extension function

- Resource vector $\mathbf{T} \in \mathbb{R}^n$
- Each node has a associated resource interval $[\mathbf{a}_i, \mathbf{b}_i]$
- A classical REF models change in resource from i to j:

$$\mathbf{f}_{ij}(\mathbf{T}) = \mathbf{T} + \mathbf{t}_{ij} \quad \text{or} \quad \mathbf{f}_{ij}(\mathbf{T}) = \max(\mathbf{a}_j, \mathbf{T} + \mathbf{t}_{ij})$$

- A path is feasible if for each node i there exists a resource vector $\mathbf{T}_i \in [\mathbf{a}_i, \mathbf{b}_i]$ s.th.

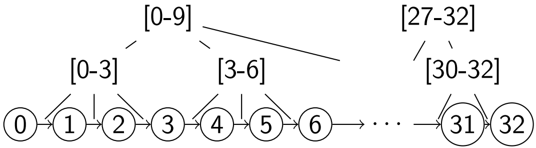
$$\mathbf{f}_{i,i+1}(\mathbf{T}_i) \leq \mathbf{T}_{i+1}$$

Segment hierarchy $\Rightarrow$ Constant time move evaluation

Aggregation:

[3-6]: $3 \to 5$, $3 \to 6$,
$4 \to 6$, inverse

[0-9]: $0 \to 6$, $0 \to 9$,
$3 \to 9$, inverse



Method

Initial solution

- Star solution: One route per customer
- Greedy solution: Nearest neighbor within capacity limit

Simple method: Local search with 3-opt move on giant tour

- Remove 3 connections/edges $\Rightarrow$ 4 parts
- Reconnect parts in all possible (new) ways $\Rightarrow$ 7 possibilities:
  $(7/6)(n-1)(n-2)(n-3)$ moves (n: #nodes in solution)
- Split in pure 3-opt and 2-opt moves:
  3-opt: $(4/6)(n-1)(n-2)(n-3)$ moves
  2-opt: $(1/2)(n-1)(n-2)$ moves

What we do on the GPU

- Once
  - Create neighborhood
- Each iteration
  - Create hierarchy
  - Evaluation of constraints and objectives for each move
  - Choosing best move

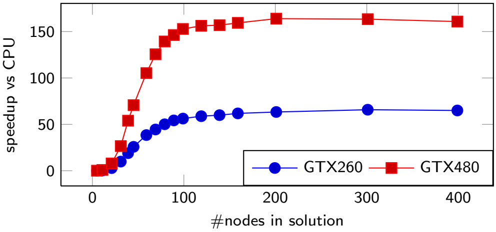
$\Rightarrow$ Neighborhood and hierarchy live whole time on GPU

What we do on the GPU

- Once
  - Create neighborhood
- Each iteration
  - Create hierarchy
  - Evaluation of constraints and objectives for each move
  - Choosing best move

$\Rightarrow$ Neighborhood and hierarchy live whole time on GPU

Both codes not optimized!

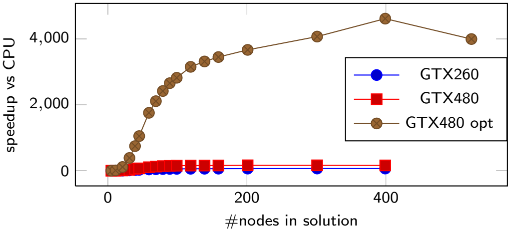


What we do on the GPU

- Once
  - Create neighborhood
- Each iteration
  - Create hierarchy
  - Evaluation of constraints and objectives for each move
  - Choosing best move

$\Rightarrow$ Neighborhood and hierarchy live whole time on GPU

Unfair comparison!
GPU is fast is known
Real task: Efficient usage
of GPU hardware

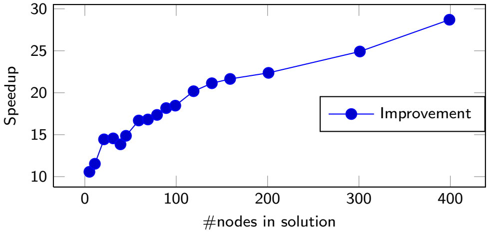


What we do on the GPU

- Once
  - Create neighborhood
- Each iteration
  - Create hierarchy
  - Evaluation of constraints and objectives for each move
  - Choosing best move

$\Rightarrow$ Neighborhood and hierarchy live whole time on GPU

More interesting:
Improvement of
GPU implementation



What to look for in the GPU implementation

Kernel execution: Kernel executes as fast as possible on GPU

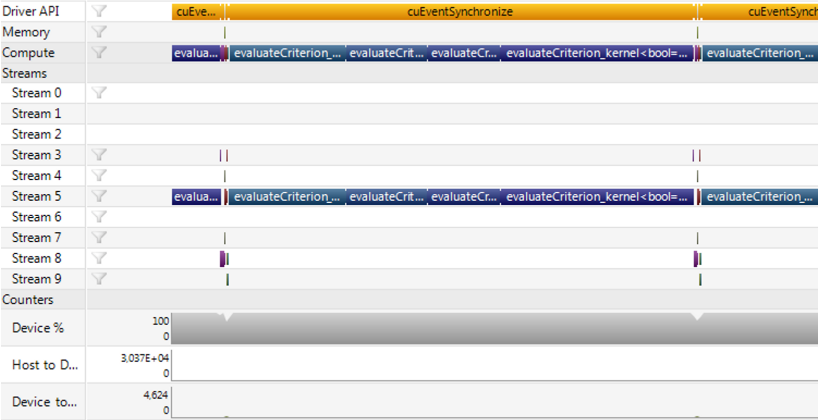
- Algorithm: Efficient for massive parallel
- Latency hiding
- Which type of memory to use
- Pattern to access memory
- Divergence in code flow $\Rightarrow$ masking
- Efficient instructions

GPU-CPU coordination: Keeping the GPU busy the whole time

- Synchronization: Asynchronous vs synchronized

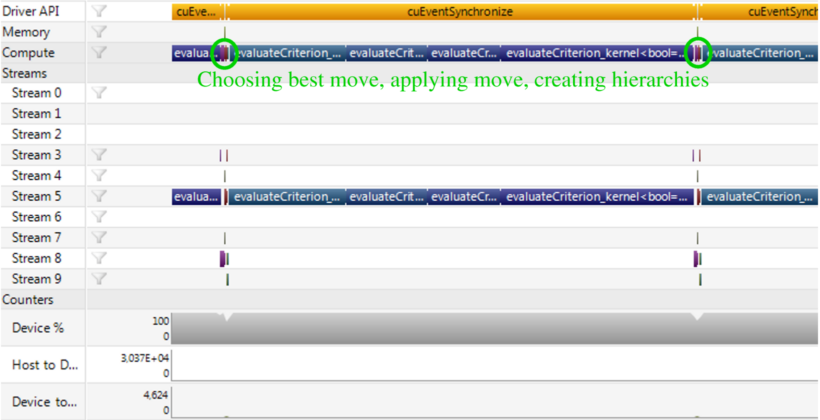
Where to Start

3-opt neighborhood for solution with 399 nodes



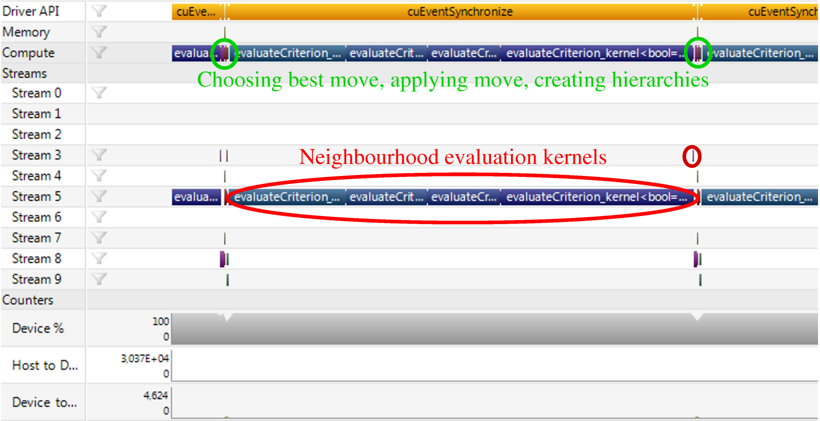
Where to Start

3-opt neighborhood for solution with 399 nodes



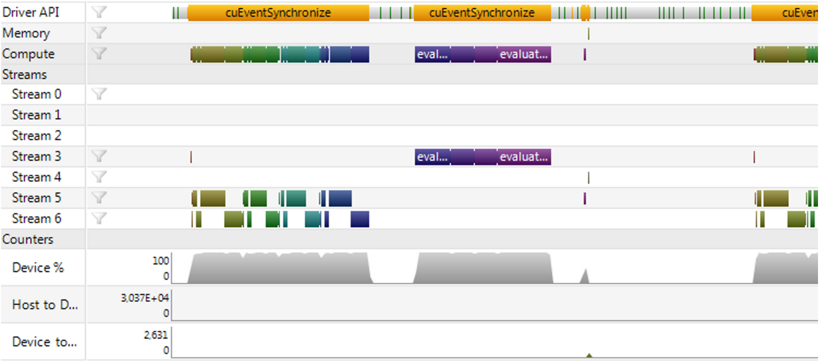
Where to Start

3-opt neighborhood for solution with 399 nodes



Where to Start

2-opt neighborhood for solution with 399 nodes



(Not so) First Prototype

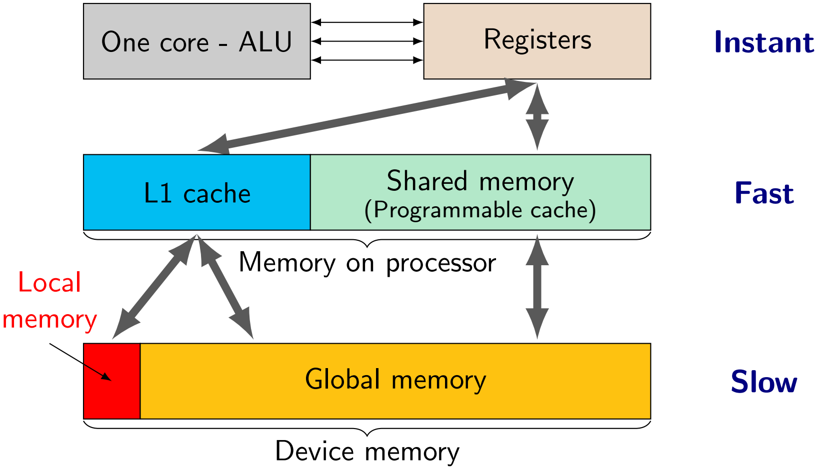
Solution with 399 nodes

Kernel		Time (ms)	Occ	Reg	Bw (Gb/s)	L1 (%)	Lm	Div
3-opt	Capacity	346	0.50	37	137	84	24e6	69e4
	Cost	241	0.67	32	140	75	10e6	68e4
	# vehicles	217	0.50	35	98	88	13e6	48e4
	Max. route length	562	0.42	47	138	75	25e6	11e5
2-opt	Capacity	0.56	0.50	36	131	80	35e3	1560
	Cost	0.38	0.50	36	117	76	15e3	1590
	# vehicles	0.36	0.50	36	93	82	20e3	1598
	Max. route length	0.85	0.42	45	129	76	37e3	2192

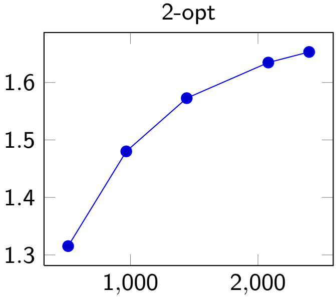
Occ: Occupancy
Bw: Bandwidth
Lm: Local memory access operations

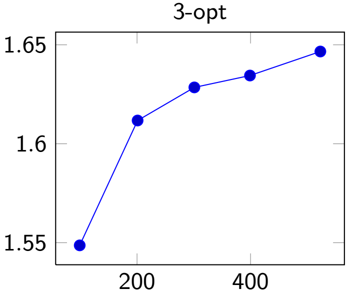
Reg: Registers
L1: L1 cache hits
Div: Divergent branches

Local memory access



Moving Data from Local Memory to Registers





x-axis: Number of nodes in solution
y-axis: Speedup

Disadvantage: Higher register usage, less occupancy

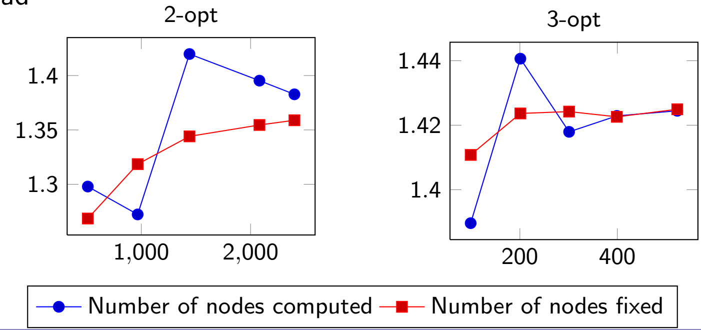
Example Efficient Instructions: Power of 2 hierarchy

Problem

Traversing hierarchy with arbitrary (but fixed) number of children per node: Modulo operations, integer division ⇒ expensive

Instead

Use only power of 2 number of children ⇒ Bitwise operations instead



Example for Memory Access Pattern: AoS vs. SoA

A hierarchy segment has 4 entries:

- Interval $[a, b]$
- Cost t
- Feasible information f

Array of Structures (AoS)

$\cdots$	b_{i-1}	t_{i-1}	f_{i-1}	a_i	b_i	t_i	f_i	a_{i+1}	b_{i+1}	t_{i+1}	$\cdots$

segment_i

Example for Memory Access Pattern: AoS vs. SoA

Array of Structures (AoS)

$\cdots$	b_{i-1}	t_{i-1}	f_{i-1}	a_i	b_i	t_i	f_i	a_{i+1}	b_{i+1}	t_{i+1}	$\cdots$

segment$_i$

Structure of Arrays (SoA)

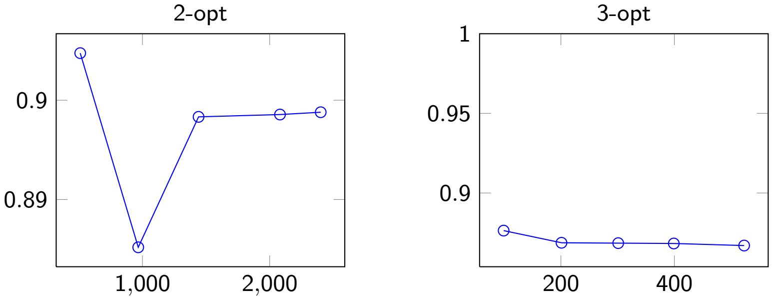
Normally:

- Neighboring threads access neighboring entries
- Better coalescing
- Fewer transactions
- Faster

$\cdots$	a_{i-2}	a_{i-1}	a_i	a_{i+1}	a_{i+2}	$\cdots$
$\cdots$	b_{i-2}	b_{i-1}	b_i	b_{i+1}	b_{i+2}	$\cdots$
$\cdots$	t_{i-2}	t_{i-1}	t_i	t_{i+1}	t_{i+2}	$\cdots$
$\cdots$	f_{i-2}	f_{i-1}	f_i	f_{i+1}	f_{i+2}	$\cdots$

segment$_i$

SoA vs AoS



x-axis: Number of nodes in solution
y-axis: Speedup of SoA against AoS

Our best kernels

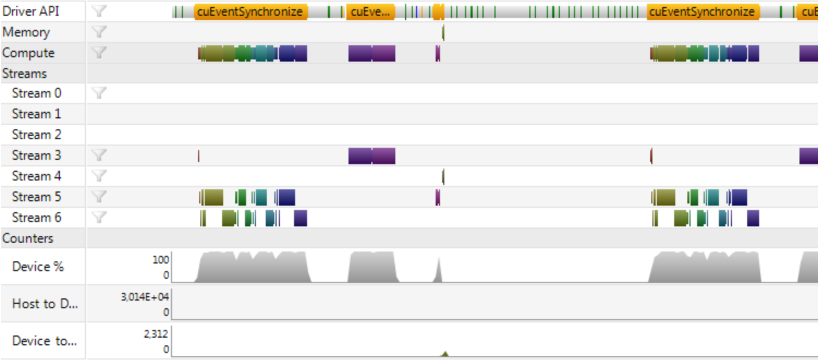
Implementing the discussed changes and other improvements leads to

Kernel		Time (ms)	Occ	Reg	Bw (Gb/s)	L1 (%)	Lm	Div
3-opt	Capacity	31	1.00	18	12	94	0	17e3
	Cost	26	1.00	20	42	77	0	2
	# vehicles	27	1.00	18	35	91	0	0
	Max. route length	76	0.52	28	12	83	0	52e4
2-opt	Capacity	0.07	1.00	18	35	83	0	31
	Cost	0.05	0.83	21	61	72	0	0
	# vehicles	0.06	1.00	18	44	84	0	0
	Max. route length	0.16	0.67	30	41	83	0	768

Occ: Occupancy
Bw: Bandwidth
Lm: Local memory access operations

Reg: Registers
L1: L1 cache hits
Div: Divergent branches

So Far: Synchronized Execution



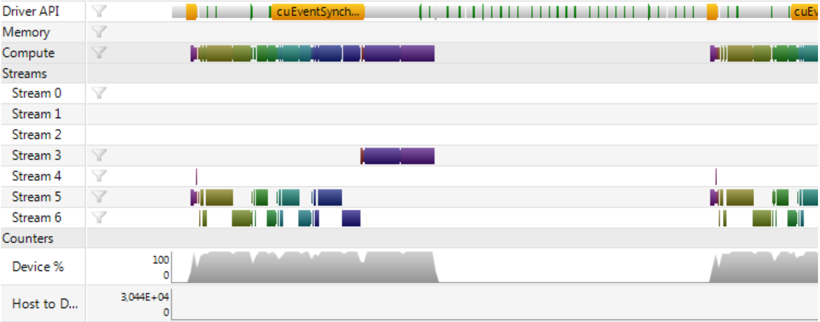
Problem: Idle GPU most of time

But:

- Can schedule GPU side reduction, move application and hierarchy creation without synchronization
- Can use hierarchy creation time to do CPU tasks

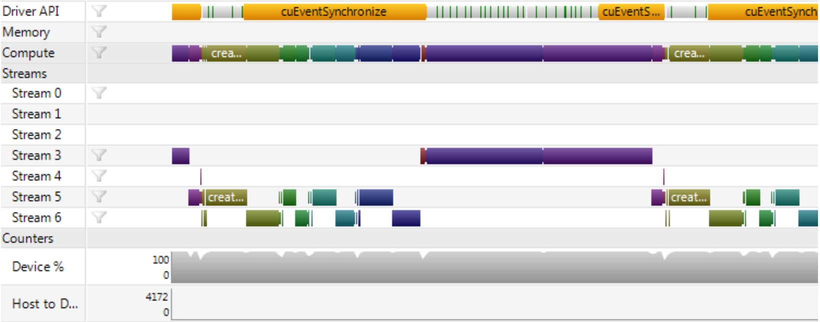
Result of Asynchronous Execution

2-opt neighborhood for solution with 399 nodes



Result of Asynchronous Execution

2-opt neighborhood for solution with 967 nodes



Works if Neighborhood large enough

Very Large Neighborhoods

Problem

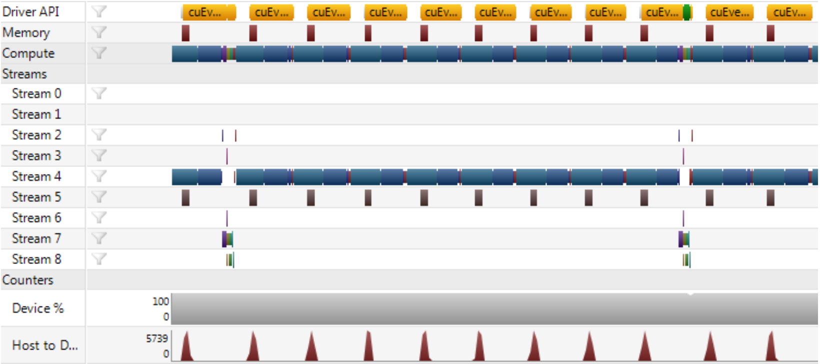
Neighborhood fitness structure too large to fit in device memory

Solution

- Split neighborhood in parts
- Evaluate each part sequentially
- Reduce two parts to size of one
- Copy neighborhood description while evaluating

Result

3-opt neighborhood for solution with 735 nodes



Quality of Solution

	Best	Our 1	Diff. 1 (%)	Our 2	Diff. 2 (%)
1	524.61	591.57	12.76	546.74	4.22
2	835.26	931.19	11.49	855.94	2.48
3	826.14	898.27	8.73	834.39	1.00
4	1028.42	1124.66	9.36	1057.89	2.87
5	1291.29	1435.18	11.14	1348.73	4.45
6	555.43	622.48	12.07	560.89	1.00
7	909.68	974.31	7.10	925.09	1.69
8	865.94	942.60	8.85	870.32	0.51
9	1162.55	1239.73	6.64	1190.80	2.43
10	1395.85	1504.12	7.76	1470.80	5.37
11	1042.11	1170.55	12.32	1046.92	0.46
12	819.56	819.56	0.00	819.56	0.00
13	1541.14	1586.69	2.96	1556.47	0.99
14	866.37	872.52	0.71	866.53	0.02

Problems from Christofides, Mingozzi, Toth

Our 1: One local search run   Our 2: Repeated local search

Summary

- Local search suited for data parallelism
- Use of GPU can lead to significant speed ups
- Challenge to get full performance of GPU
- Asynchronous execution to keep GPU busy at all times
- Very large neighborhoods possible

Thank you for your attention!