Using Heterogeneous Computing for Solving Vehicle Routing Problems

GPU based local search for CVRP with REFs

Geir Hasle, Oddvar Kloster, Atle Riise, Christian Schulz, Morten Smedsrud
SINTEF ICT

21. June 2010

Outline

Motivation

Transportation management

- Goal: good solutions computed fast, based on thorough exploration of search space
- Increase in computing power $\Rightarrow$ existing methods faster or more exploration
- Better algorithms, methodological improvements

Variety of methods for solving VRP

- Metaheuristics
- Heuristics based on exact methods and hybrid methods
- Variants and hybrids of large neighbourhood search
- Variable neighbourhood search
- Iterated local search

Motivation cont'd

Parallelism often occurs naturally in the methods

- Algorithmic level, metaheuristics
- Iteration level, neighbourhood evaluation (generation)
- Solution level

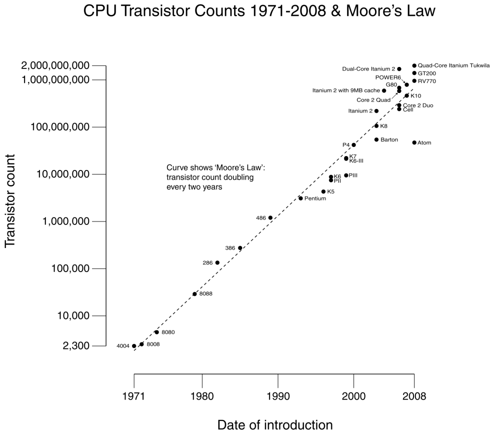
Parallel platforms

- Traditional supercomputers: Cluster (large number) of CPUs
  High level of independence, can perform basically independent tasks $\Rightarrow$ Task parallelisation
- Parallel methods in optimisation not new, but most focus on task parallelisation (according to Crainic 2008)
- What about the new multi-core CPUs
- What about the GPUs

Moore's law

The number of transistors that can be placed inexpensively on an integrated circuit doubles every two years.

- 1971 (4004),
  2300 transistors,
  1 x 0.000740 GHz
- 2004 (Pentium 4 Prescott),
  125 000 000 transistors,
  1 x 4 GHz
- 2008 (Core i7 Quad),
  731 000 000 transistors,
  4 x 3.33 GHz



Picture from http://en.wikipedia.org/wiki/Moores_law

What happened?

Increasing frequency hits three major problems (walls): Memory, ILP, Power density (power/area)

Memory

- Memory speeds did not increase as fast as core frequencies
  Processor can wait hundreds of clock cycles for data/instructions from main memory
- Wait can be reduced by larger caches and instruction level parallelism

Instruction level parallelism

- Difficult to find enough parallelism in instructions stream of single process to keep cores busy

Multi-core

Power density (heat)

- Increase in frequency leads to increase in power density
- CPU has higher power density than a cooking plate
- Using about 80% of frequency halves power consumption

$\Rightarrow$ Use of 2 cores with $\sim$ 80% of frequency: same power consumption, $\sim$ 160% performance

But: Deep pipelines, heavy ILP use and huge caches drain a lot of power

$\Rightarrow$ no 100 core processor

Acceleration cores

- Shallow pipelines, low or no ILP, small or no caches
- Power efficient

Heterogeneous computer

Classical supercomputer consist of many processors, maybe with dual/quad core
⇒ Consume lot of power, maintenance, expensive

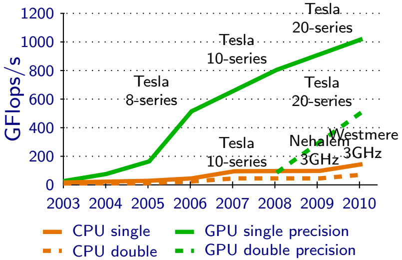
But: Commodity PCs nowadays have multi-core CPUs and one (or more) GPU (has acceleration cores)
⇒ Cheap, high performance if it can be harnessed

Heterogeneous computer: Tightly coupled system of processing units with distinct characteristics

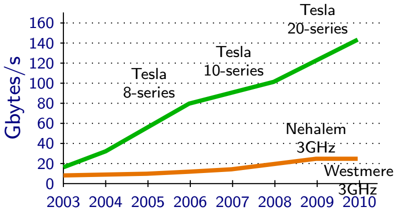
GPU

- Background: Computer graphics
- Nowadays: General purpose GPU
- Massively parallel: 512 cores
- High memory bandwidth
- Typical speedup: 10-50 (to CPU)
- Data parallelism: Typically same task performed by each core on different pieces of data
- NVIDIA Fermi:
  - IEEE 754-2008 floating point standard
  - Improved double precision performance (now half of single precision)

Peak float performance



Peak memory bandwidth



Programming GPU

Direct Compute

- Part of Microsoft DirectX
- Debugger (NVIDIA) on Windows

OpenCL (AMD, NVIDIA)

- Extension of C, reminiscent of GLSL
- Relatively immature, but improves as we speak

Cuda (NVIDIA)

- Large subset of C++, can share code with CPU code
- Mature
- Debugger on Linux and Windows

GPU in Science

GPU usage in other Sciences/Industry

- PDE / Simulation: Shallow water
- Medicine: Automated ultrasound imaging system
- Finance: Analyses the entire U.S. equity options market in real time

GPU in Optimisation

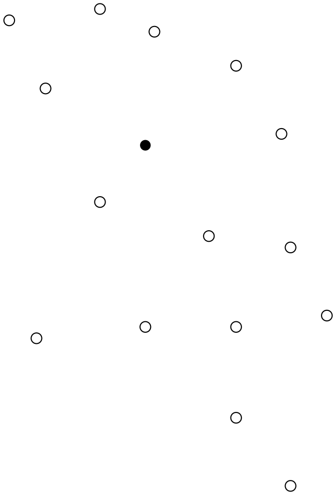
- Knapsack
  M. Scherger, Two Parallel Algorithms to Solve the 2D Knapsack Problem Using GPUs, 2008
  D. M. Quan, and L. T. Yang, Solving 0/1 Knapsack Problem for Light Communication SLA-Based Workflow Mapping Using CUDA, 2009
- Evolutionary algorithms
  Harding, S. and W. Banzhaf, Fast Genetic Programming on GPUs, 2007
  Langdon, W. and W. Banzhaf, A SIMD Interpreter for Genetic Programming on GPU Graphics Cards, 2008
- Neighbourhood evaluation
  Janiak, A., W. Janiak, and M. Lichtenstein, Tabu Search on GPU, 2008
  Luong, T.V., N. Melab, and E.-G. Talbi, Parallel Local Search on GPU, 2009

$\Rightarrow$ Good point in time to start using it

Capacitated Vehicle Routing Problem

Given:

- A depot and number of customer nodes
- Length/Cost c_{ij} between nodes
- Capacity of vehicle(s) C
- Demand of customers $d_i \leq C$



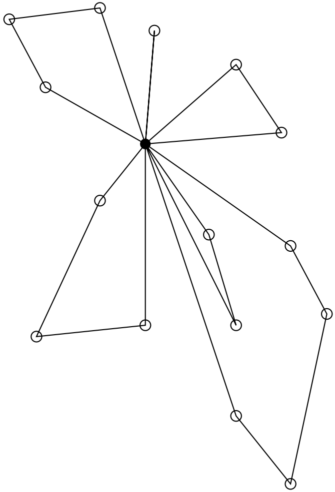
Capacitated Vehicle Routing Problem

Given:

- A depot and number of customer nodes
- Length/Cost c_{ij} between nodes
- Capacity of vehicle(s) C
- Demand of customers $d_i \leq C$

Wanted: Route(s)

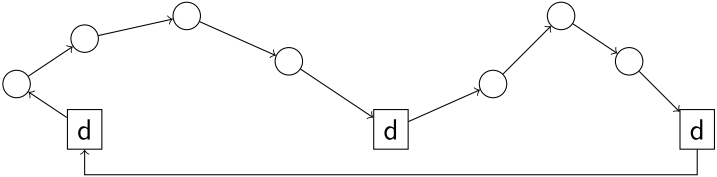
- Each customer is visited once
- Each route visits depot
- Minimal length/cost
- Capacity feasible



Model

Used 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

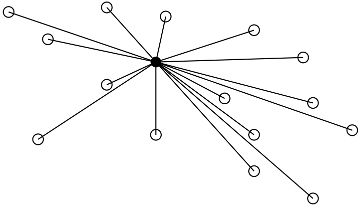


- Use of classical resource extension functions to model capacity constraint

Simple method: Local search with 3-opt move

Initial solution

- Star solution: A single route to each customer



3-opt move

- Remove 3 connections/edges ⇒ 4 segments
- Reconnect in all possible ways ⇒ 7 possibilities
  $1-3-2-4$, $1-3-\bar{2}-4$, $1-\bar{3}-2-4$, $1-\bar{3}-\bar{2}-4$, $1-2-\bar{3}-4$, $1-\bar{2}-3-4$, $1-\bar{2}-\bar{3}-4$

⇒ Nearly $(7/6)(n-1)(n-2)(n-3)$ moves
(n: number of nodes in solution)

Classical Resource extension function

Resource constraints modeled by resource consumption

- Resource: cost, time, load, distance, ...
- Resource vector $\mathbf{t} \in \mathbb{R}^n$
- Each node has a associated resource interval $[\mathbf{a}_i, \mathbf{b}_i]$
- Change of resource consumption from i to j: $\mathbf{f}_{ij} : \mathbb{R}^n \rightarrow \mathbb{R}^n$
- 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 REF:

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

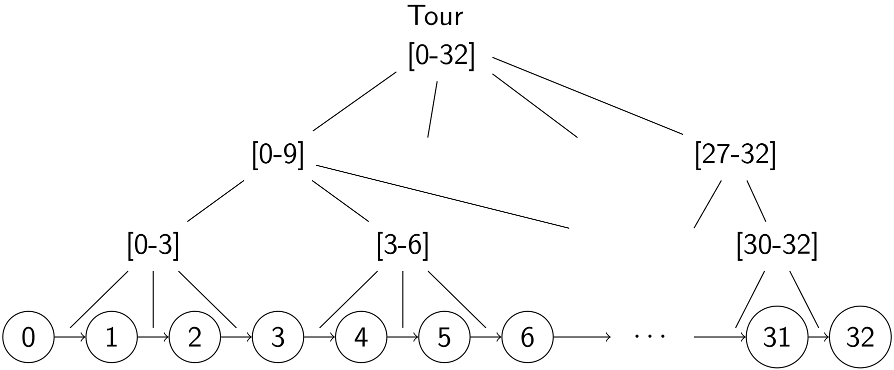
CVRP (capacity): Classical REF with

$a_i = 0, \qquad b_i = C,$

$t_{ij} = d_j$ for j a customer, $t_{ij} = -C$ for j depot

Segment - Hierarchy

Why classical REF? Simple, can build segment hierarchy



Aggregation:

[3-6] contains: $3 \rightarrow 5$, $3 \rightarrow 6$ and $4 \rightarrow 6$ and inverse

[0-9] contains: $0 \rightarrow 6$, $0 \rightarrow 9$ and $3 \rightarrow 9$ and inverse

Segment - Hierarchy cont'd

- Why segment hierarchy? Gives constant time feasibility check

  Example: Exchange two nodes, e.g. 5 and 20:

path up to first:	$0 \rightarrow 4$:	$0 \rightarrow 3$, $3 \rightarrow 4$
reconnect first:	$4 \rightarrow 20$:	
	$20 \rightarrow 6$:	
path to second:	$6 \rightarrow 19$:	$6 \rightarrow 9$, $9 \rightarrow 18$, $18 \rightarrow 19$
reconnect second:	$19 \rightarrow 5$:	
	$5 \rightarrow 21$:	
path to end:	$21 \rightarrow 32$:	$21 \rightarrow 27$, $27 \rightarrow 32$

- Maximum number of segments in one path: 2l-1 (l: depth of hierarchy)
- How to do feasablity check with segments, see paper(s) by Irnich
- Effort to create hierarchy: $O(n^{2^l/(2^l-1)})$

Parallel local search

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

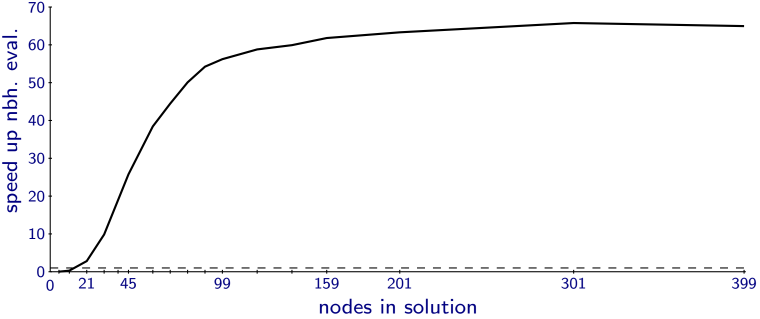
What we do on the GPU

Transfer of data GPU $\leftrightarrow$ CPU slow $\Rightarrow$ try to minimize/avoid it

On GPU

- Once:
  - Create neighbourhood
- Each iteration:
  - Create hierarchy
  - Evaluation of capacity constraint and length objective for each move
  - Choosing best move
- Neighbourhood and hierarchy live whole time on GPU, no transfer
- Transfer once: constraint & objective data
- Transfer per iteration: move, solution (for now)

Does it pay?



Early timing, only gives indication:

- CPU code is not optimized
- GPU code is not optimized

GPU is fast is known, real task: Efficient usage of GPU hardware

Why optimize GPU code

Example reduction, taken from NVIDIA CUDA SDK whitepaper

	Time (2^{22} ints)	Bandwidth	Step Speedup	Cumulative Speedup
Kernel 1: interleaved addressing with divergent branching	8.054 ms	2.083 GB/s		
Kernel 2: interleaved addressing with bank conflicts	3.456 ms	4.854 GB/s	2.33x	2.33x
Kernel 3: sequential addressing	1.722 ms	9.741 GB/s	2.01x	4.68x
Kernel 4: first add during global load	0.965 ms	17.377 GB/s	1.78x	8.34x
Kernel 5: unroll last warp	0.536 ms	31.289 GB/s	1.8x	15.01x
Kernel 6: completely unrolled	0.381 ms	43.996 GB/s	1.41x	21.16x
Kernel 7: multiple elements per thread	0.268 ms	62.671 GB/s	1.42x	30.04x

Local Search with 3opt - Results

Problem	Our	Best	#It.	Time(s)	Nbh size
P-n16-k8	473.782	451.335	7	0.109	28420
P-n20-k2	233.995	217.416	18	0.327	59052
P-n23-k8	560.598	531.174	13	0.281	92708
E-n30-k3	508.139	535.797	31	1.013	215992
B-n35-k5	1403.96	956.294	32	1.432	350812
P-n40-k5	506.039	461.726	37	2.290	532532
F-n45-k4	727.746	723.541	43	3.598	768152
B-n50-k7	745.160	744.228	44	4.890	1064672
A-n60-k9	1407.09	1355.800	56	10.731	1868412
P-n70-k10	915.380	829.933	60	18.301	2999752
A-n80-k10	1833.49	1766.500	75	34.391	4514692
E-n101-k8	990.737	828.737	97	90.523	9193800
M-n151-k12	1124.44	1043.410	144	475.321	31185700
M-n200-k16	1402.67	1499.780	190	1585.751	72998772

Summary & Future Work

Summary

- Your office PC is a heterogeneous computer
- Proper algorithms can harness CPU+GPU power
- Early results in local search for CVRP promising

Future Work

- Optimise code
- Larger solutions: memory, number of tasks
- More advanced strategies such as metaheuristics
- Keep CPU and GPU busy

Thank you for your attention!