THE UNIVERSITY OF RHODE ISLAND

A MULTI-GPU PARALLEL GENETIC ALGORITHM FOR LARGE-SCALE VEHICLE ROUTING PROBLEMS

Marwan Abdelatti[1], Manbir S. Sodhi[1], and Resit Sendag[2]

[1]Department of Industrial And Systems Engineering.
[2]Department of Electrical And Computer Engineering.

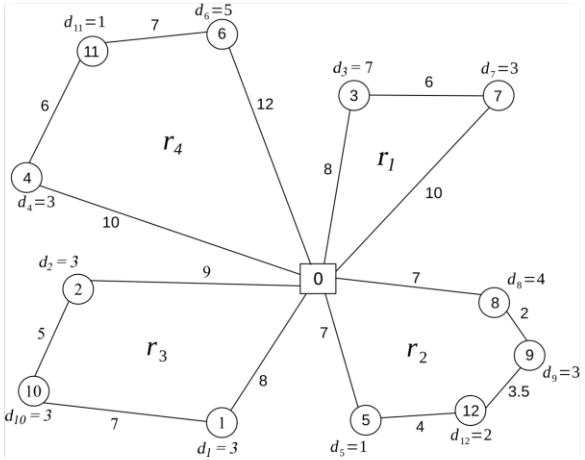
University of Rhode Island

mabdelrazik@uri.edu, sodhi@uri.edu, sendag@uri.edu

INTRODUCTION

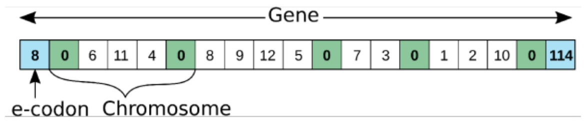
- The Vehicle Routing Problem (VRP) is fundamental to logistics planning.
- It seeks the optimal set of routes for a fleet of trucks to serve a given set of customers subject to some constraints.
- There are many forms of VRP for example: Dynamic VRP (DVRP), VRP with Time Windows VRPTW, and Capacitated VRP (CVRP).
- CVRP is the most studied form of VRP where:

    1. Total customer demands cannot exceed the truck capacity.
    2. Customers must be visited only once.



MOTIVATION

- GAs are one of the methods commonly used to solve VRP.



Special property: they operate on a population of potential solutions:

Improves algorithm efficiency, and the probability of finding a good solution.

Get stuck in local minima, longer time to converge with large problems

High Computational Cost!

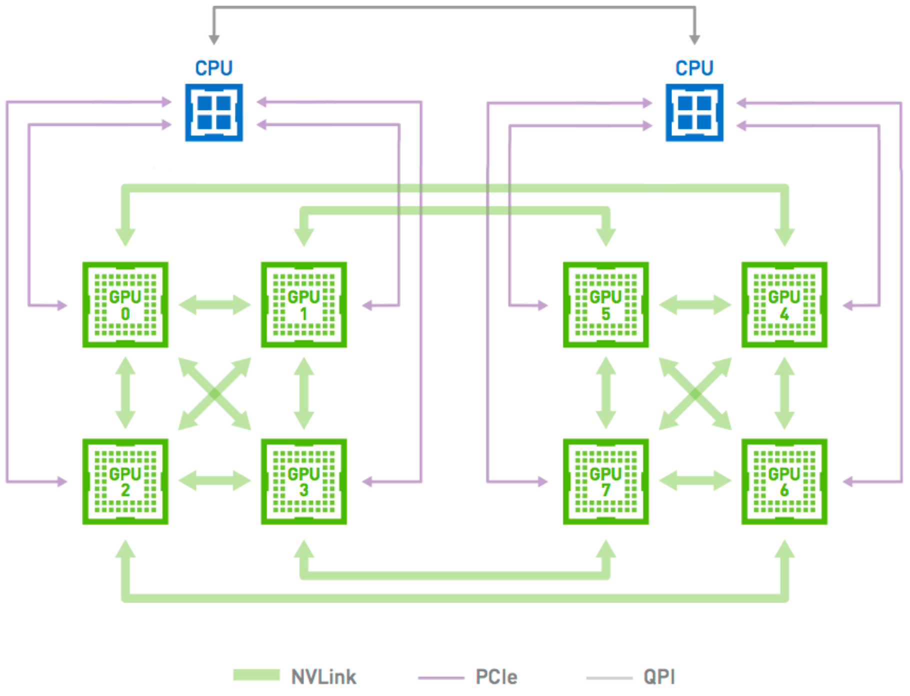
HARDWARE

The algorithm runs entirely on a DGX-1 server:

- 8 GPUs (NVIDIA Tesla v100): each has 32 GB shared memory and 5,120 CUDA cores.
- Hybrid cube-mesh topology through NVLink.

Benefits of using NVLink and P2P:

- Direct data flow between the GPUs.
- Relieves the pressure on PCIe bus, the CPUs, and the system memory.
- Higher bandwidth and lower latency than PCIe links.



Indirect communications might cause delays!!

RESULTS & DISCUSSION (SETTINGS)

- The execution speed in $secs/generation$ is reported on selected benchmark problems of size between 420 and $20{,}000$ nodes.
- GA parameters are taken from a previous design of experiment (DOE):
  - Population size: $20\times$ the number of nodes n,
  - Inverse mutation at a 0.3 probability,
  - 1-point crossover at a 0.6 probability,
  - Migration rate of $1/GPU\_count$ of it each GPU population,
  - Migration interval of $1{,}000$ generations, and
  - The demes are not synchronized at the migration time to avoid further delays
- 4 GPU arrangements were considered $(1, 2, 4 \,\&\, 8)$ as well as one parallel CPU algorithm. Each arrangement run each problem 5 times for $5{,}000$ generations.

RESULTS & DISCUSSION

Instance	Node Count	CPU	GPU arrangement			
			1	*2*	*4*	*8*
X-n420-k130.vrp[a]	420	360	0.78	0.5	0.46	**0.4**
Li_30.vrp[b]	1,040	1,800	18.67	9.03	4.71	**1.05**
Li_31.vrp[b]	1,120	X*	12.18	6.11	3.18	**1.22**
Li_32.vrp[b]	1,200	X*	9.92	4.89	2.67	**1.80**
33.vrp[c]	2,401	X*	80.28	29.17	12.25	**6.71**
34.vrp[c]	3,601	X*	253.94	145.51	54.62	**21.47**
35.vrp[c]	6,001	X*	1,067.1	1,062.6	397.89	**163.68**
Ghent1.vrp	10,000	X*	X*	X*	X*	**1103.8**
Flanders1.vrp	20,000	X*	X*	X*	X*	**2318.9**

* The problem is too big for this arrangement

- The algorithm was run for 100,000 generations on the 420-node problem and got a solution gap of 4.98% from the best-known solution in the literature.

RESULTS & DISCUSSION (PROFILING)

- Profiling was performed on a problem 6,001 nodes:

Running [/home/[redacted]/nsight-systems-2022.1.1/target-linux-x64/reports/cudaapisum.py Downloads/prof_Ky_35_8gpus.sqlite]...

Time (%)	Total Time (ns)	Num Calls	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
99.0	333,792,031,463,153	134,714	2,477,782,795.1	4,605.0	2,299	39,745,047,666	6,771,501,868.2	cuStreamSynchronize
0.4	1,341,112,616,325	11,251,823	119,190.7	20,027.0	16,088	423,340,776	2,373,310.5	cudaMemcpy
0.3	857,874,468,117	229	3,746,176,716.7	352,112.0	195,902	31,071,026,971	8,689,844,668.8	cuModuleLoadDataEx
0.2	515,423,467,827	11,948,576	43,136.8	9,324.0	3,853	23,690,688,585	8,399,554.1	cuLaunchKernel
0.1	374,808,831,188	47	7,974,655,982.7	4,384,067,599.0	43,899,688	29,207,972,911	8,047,133,084.2	cudaMemcpyPeer

CUDA API

Running [/home/[redacted]/nsight-systems-2022.1.1/target-linux-x64/reports/gpukernsum.py Downloads/prof_Ky_35_8gpus.sqlite]...

Time (%)	Total Time (ns)	Instances	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
42.3	141,295,611,300,048	4,810	29,375,386,964.7	29,351,465,547.5	5,730,032,465	39,734,140,897	2,126,856,633.6	cudapy::kernels::find_duplicates…
28.5	95,280,965,906,390	4,805	19,829,545,454.0	19,740,856,891.0	15,306,774,983	29,655,096,678	2,017,849,756.1	cudapy::kernels::twoOpt$2473(Arr…
14.3	47,767,864,922,369	4,803	9,945,422,636.3	8,856,488,169.0	2,793,094,917	16,298,794,407	3,573,339,843.0	cudapy::kernels::addMissingNodes…
12.9	43,086,587,684,137	4,809	8,959,573,234.4	8,932,402,511.0	8,377,041,422	23,896,849,953	438,697,471.9	cudapy::kernels::shift_r_flag$24…
1.6	5,356,944,563,293	4,809	1,113,941,477.1	1,109,480,248.0	991,568,120	1,285,318,460	44,007,499.3	cudapy::kernels::cap_adjust$2419…

GPU Kernels

RESULTS & DISCUSSION (PROFILING)

Running [/home/[redacted]/nsight-systems-2022.1.1/target-linux-x64/reports/gpumemsizesum.py Downloads/prof_Ky_35_8gpus.sqlite]...

Total (MB)	Count	Avg (MB)	Med (MB)	Min (MB)	Max (MB)	StdDev (MB)	Operation
2,540,209.342	156,100	16.273	0.036	0.000	540.180	79.145	[CUDA memcpy DtoD]
25,400.073	11,251,871	0.002	0.000	0.000	540.180	1.104	[CUDA memcpy DtoH]
25,389.228	55	461.622	540.180	0.096	540.180	192.166	[CUDA memcpy HtoD]
23,263.656	50	465.273	540.180	72.012	540.180	173.376	[CUDA memset]

Memory Transfer (size)

Running [/home/[redacted]/nsight-systems-2022.1.1/target-linux-x64/reports/gpumemtimesum.py Downloads/prof_Ky_35_8gpus.sqlite]...

Time (%)	Total Time (ns)	Count	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Operation
69.5	20,941,828,654	11,251,871	1,861.2	1,632.0	1,152	57,213,067	106,605.3	[CUDA memcpy DtoH]
22.2	6,693,209,423	156,100	42,877.7	1,248.0	1,023	1,396,120	202,930.2	[CUDA memcpy DtoD]
8.1	2,450,018,745	55	44,545,795.4	51,115,037.0	11,679	57,218,142	18,639,170.6	[CUDA memcpy HtoD]
0.1	26,017,855	50	520,357.1	603,691.0	82,688	605,405	192,822.3	[CUDA memset]

Memory Transfer (time)

CONCLUSIONS

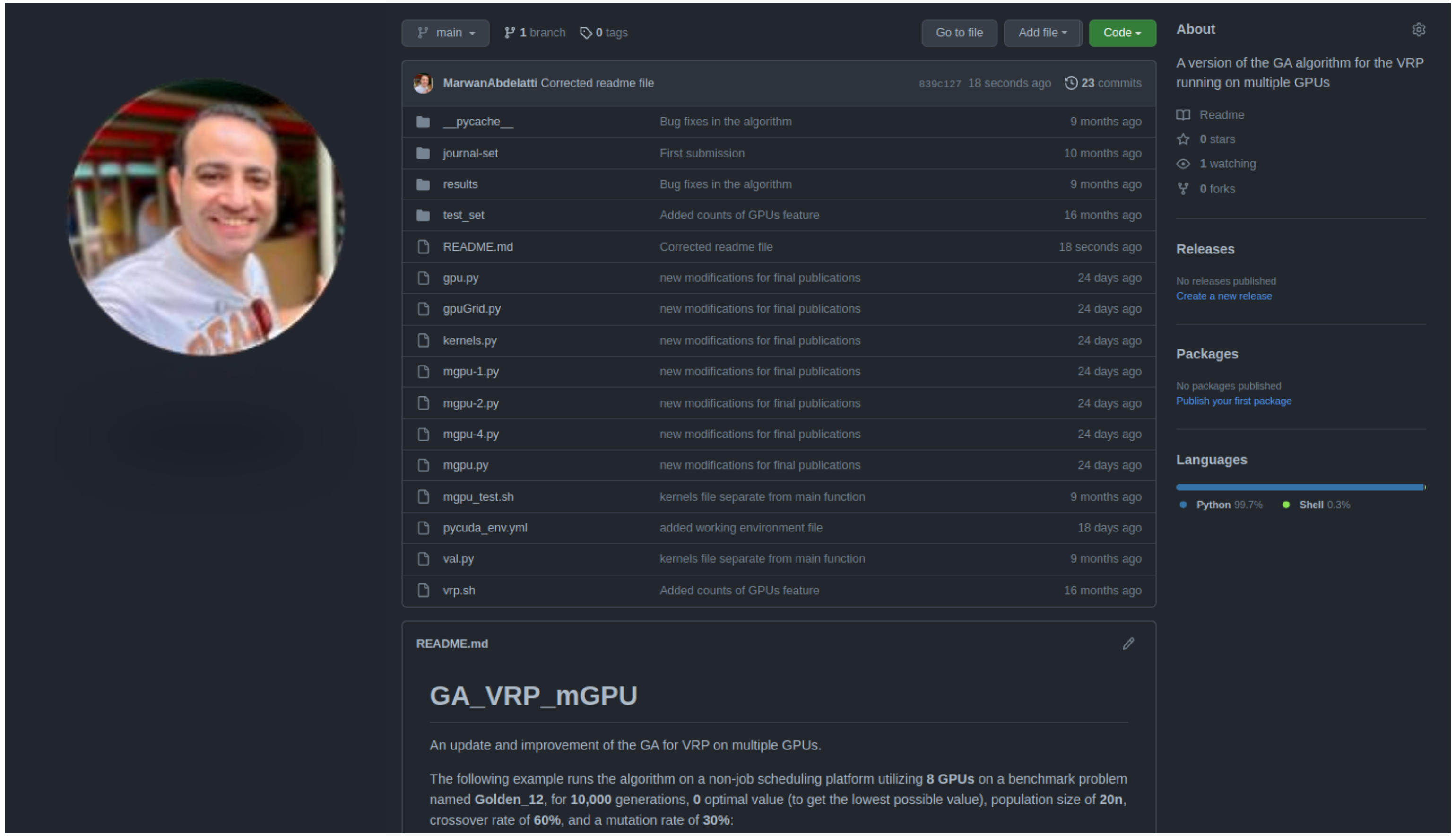
- We utilize multiple GPUs for a real-world problem that directly impacts logistics operations.
- GA and 2-opt local search are utilized for large-scale CVRP.
- Tested on different hardware arrangements (1, 2, 4, and 8 GPUs) and SIMD parallel CPU implementation.
- Execution speeds and profiling show that multiple GPUs have significant improvements over CPU or single-GPU utilization despite the communication lags between GPUs.
- We obtained a high-quality solution compared with the best-known solution in the literature for a problem of choice.

FUTURE WORK

- This implementation is explicit to a specific communication topology and P2P support.
- Future work will include improvements to execute on multi-GPU clusters with varying connection topologies.
- Utilization of shared memory and capitalization of GPU tensor cores for math operations.

CODE CLONING

https://github.com/MarwanAbdelatti/GA_VRP_mGPU



REFERENCES

[1] Samuel Eilon, Carl Donald Tyndale Watson-Gandy, Nicos Christofides, and Richard de Neufville. 1974. Distribution management-mathematical modelling and practical analysis. *IEEE Transactions on Systems, Man, and Cybernetics* 6 (1974), 589–589.

[2] Geoff Clarke and John W Wright. 1964. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research* 12, 4 (1964), 568–581.

[3] Michel Gendreau, Alain Hertz, and Gilbert Laporte. 1994. A tabu search heuristic for the vehicle routing problem. *Management science* 40, 10 (1994), 1276–1290.

[4] Jean-Yves Potvin and Jean-Marc Rousseau. 1995. An exchange heuristic for routing problems with time windows. *Journal of the Operational Research Society* 46, 12 (1995), 1433–1446.

[5] Barrie M Baker and MA Ayechew. 2003. A genetic algorithm for the vehicle routing problem. *Computers & Operations Research* 30, 5 (2003), 787–800.

[6] Abel Garcia-Najera and John A Bullinaria. 2009. Comparison of similarity measures for the multi-objective vehicle routing problem with time windows. *In Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. 579–586.

[7] Christian Prins. 2004. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research* 31, 12 (2004), 1985–2002.

[8] Duane Storti and Mete Yurtoglu. 2015. *CUDA for engineers: an introduction to high-performance parallel computing*. Addison-Wesley Professional.

[9] Jason Sanders and Edward Kandrot. 2010. *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional.

QUESTIONS