Trial lecture

Discrete Optimization on GPUs (Graphics Processing Units)

Jianyong Jin

Molde University College, Specialized University in Logistics, Norway

March 22, 2013

Outline

Discrete optimization

- The term **discrete optimization** is often used to describe the class of optimization problems that can be solved by a search through a finite set of variants(Leont'ev, 2007).
- The variables used in the mathematical model are restricted to assume only discrete values.
- Opposed to continuous optimization where the variables can assume real values.

Discrete optimization problem (DOP)

- A DOP is either a minimization or maximization problem.
- A mathematical formulation of a minimization problem:

$$\min_{x \in S} \; f(x)$$

- The objective function $f(x)$ is defined on a discrete set.
- A finite set of feasible solutions S.
- The goal is to find the optimal solution whose objective function value is better than all other feasible solutions.
- DOPs are often computationally hard.

Some discrete optimization problems

- Integer linear programming.
- Knapsack problem.
- Assignment problem.
- Traveling salesman problem.
- Vehicle routing problem.
- Minimum spanning tree problem.
- Location problem.

Solution methods

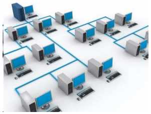
- Exact algorithms
  - Able to give the optimal solution.
  - Often can not solve large instances within a reasonable computing time.
- Approximate algorithms
  - Do not guarantee to find the optimal solution.
  - There is a bound on the solution quality.
- Heuristic algorithms
  - Do not guarantee to find the optimal solution.
  - Do not have a bound on the solution quality.
  - In practice, heuristic algorithms, especially metaheuristics, have been proved successful (Leont'ev, 2007).

Metaheuristics

- Local search (LS) based
  - Start from an initial solution.
  - Move to another solution in the neighborhood of the current solution iteratively.
  - Tabu search, simulated annealing, etc.
- Population based
  - Maintain a population of solutions.
  - Replace some members of the population iteratively.
  - Genetic algorithm, etc.
- Learning mechanisms based
  - Apply learning mechanism during the search.
  - Ant colony optimization, etc (Laporte, 2007).

Parallel algorithms

- For solving large or complex DOPs, parallel (exact or heuristic) algorithms have been increasingly used.
- Parallel algorithms are executed by multiple processes (threads) simultaneously on multiple processors in order to solve a given problem.
- Opposed to sequential algorithms that are executed sequentially on a single processor.



Goals of parallel algorithms

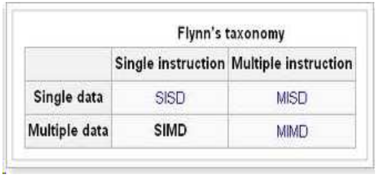
- Speed up the search.
- Improve the quality of the solutions obtained.
- Improve the robustness of the algorithm.
- Solve large scale problems (Crainic and Toulouse, 2010).

Major parallelization forms

- **Functional parallelism**: different tasks are allocated to different processors.
- **Data parallelism (domain decomposition)**: different processors perform the same task on sub-problems related to different data.
- **Algorithm parallelism**: multiple algorithms are run in parallel (Crainic and Toulouse, 2010) .

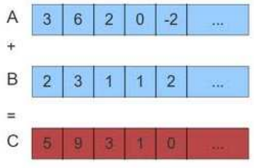
Parallel computers

- Multi-core computers.
- Multi-computers, e.g. clusters, grids.
- Specialized parallel computer architectures (e.g. Graphics processing units (GPUs)) together with traditional processors.



Flynn's taxonomy

	Single instruction	Multiple instruction
Single data	SISD	MISD
Multiple data	SIMD	MIMD



GPU computing

- Used in the past only for graphics and video applications.
- Lately have been increasingly used for scientific computing.
- Release of the programming tools that enable the use of GPUs for general purpose computing.
- Easily accessible, large speedup.
- GPU computing: the use of a GPU together with a CPU to accelerate general-purpose scientific and engineering applications.
- Also known as General-Purpose computation on GPUs (GPGPU) (Brodtkorb et al., 2013).

Major GPU vendors

Vendor	Market
NVIDIA	high-performance and discrete graphics
AMD	high-performance and discrete graphics
Intel	integrated and low-performance

Source: Brodtkorb et al., (2013).

Warp and wavefront

- The most basic unit of scheduling of the GPU.
- The minimum size of the data processed in SIMD fashion.
- For NVIDIA GPUs, threads execute in 32-thread groups called warps.
- For AMD GPUs, threads execute in 64-thread groups called wavefronts.
- Divergence in a warp (wavefront) causes sequential execution, and should be avoided.

CPU vs GPU

- CPU
  - Several arithmetic and logic units, a large cache and a control unit.
  - Manage multiple and different tasks requiring lots of data.
- GPU
  - A large number of arithmetic units with a limited cache and few control units.
  - High memory bandwidth.
  - High floating point performance.
  - Address problems that can be expressed as data-parallel computations (Luong, 2012).

Molde University College
Specialized University in Logistics

General GPU computing model

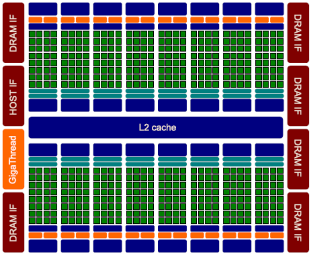
- CPU works as a host and GPU is used as a device coprocessor.
- Data-parallel computations can be assigned to GPU.
- Data must be transferred between CPU and GPU.
- Processors on GPU support the single program multiple data (SPMD) fashion.
- *Kernel* is used to define the programs for GPU processors.
- A kernel is a function callable from CPU and executed on GPU simultaneously by multiple processors on different data(Luong, 2012).

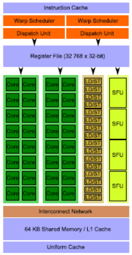
Programming language for GPU

For general-purpose computing:

- Open Computing Language (**OpenCL**).
- Microsoft **DirectCompute**.
- Compute Unified Device Architecture (**CUDA**): created by NVIDIA, is the most mature technology with the most advanced development tools (Brodtkorb et al., 2013).

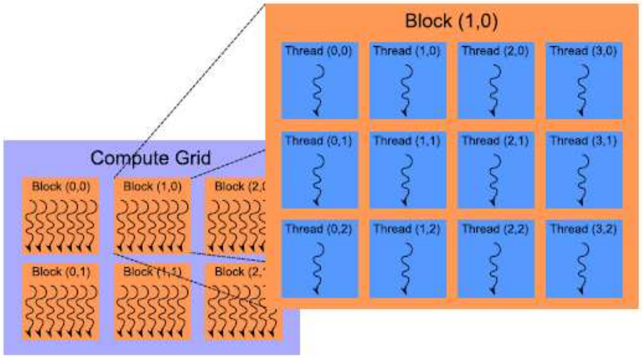
NVIDIA GPU architecture





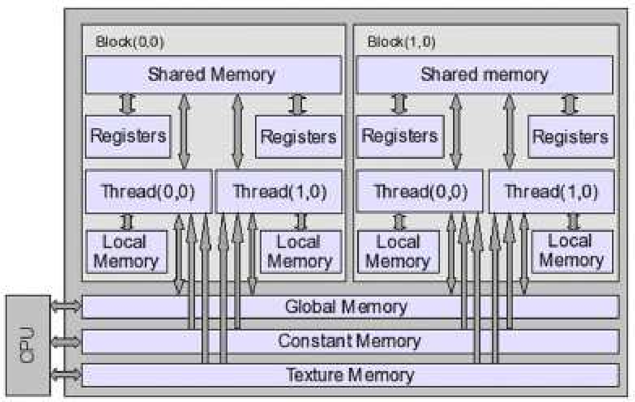
16 streaming multiprocessors (SMs), each SM has 32 streaming processors(SPs). In total, 512 SPs.

CUDA thread organization



CUDA thread organization

- Launching a kernel creates a grid of threads.
- The threads are grouped into blocks, and the blocks constitute the grid.
- The threads within a block can synchronize and cooperate.
- A block runs on a single multiprocessor.
- Each block has an index within the grid, each thread has an index within its block.
- Each thread uses its block index and its thread index to identify its position in the global grid (Brodtkorb et al., 2013).



CUDA memory model

CUDA memory model

Memory	Usage	Scope	Latency	Size
Registers	R/W	Thread	Very fast	Very small
Local	R/W	Thread	Medium	Medium
Shared	R/W	Block	Fast	Small
Global	R/W	Grid+CPU	Medium	Big
Constant	Read only	Grid+CPU	Fast	Medium
Texture	Read only	Grid+CPU	Fast	Medium

Source: Kirk and Hwu (2010), Luong (2012).

CUDA to OpenCL

CUDA	**OpenCL**
Grid	NDRange
Block	Work-group
Thread	Work-item
Registers	Private memory
Shared memory	Local memory
Global memory	Global memory
Constant memory	Constant memory
Texture memory	Image memory

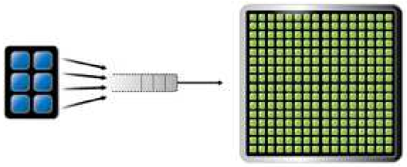
Source: Martinez et al. (2011)

Main design issues

- Task partition between CPU and GPU.
- Parallelism control.
- Memory management (Luong, 2012).

Task partition

- Select suitable tasks for GPU.
- Optimize data transfer between CPU and GPU because it is time consuming.



Parallelism control

- Control the number of threads to satisfy memory constraints.
- Have enough threads to obtain the best multiprocessor occupancy and to hide memory latency.
- Use thread ID to map each thread to a specific piece of data(Luong, 2012).
- Consider cooperation among threads within a block.

Memory management

- Consider memory constraints when defining kernels.
- Minimize access to global memory.
- Copy one part of global memory to a shared memory.
- Global memory access coalescing: if threads in a warp can access consecutive memory addresses, all the access can be combined into a single request (Kirk and Hwu, 2010).

Speedup

Normally

$$Speedup = \frac{\textit{the running time of the sequential version of the algorithm}}{\textit{the running time of the parallel version of the algorithm}}$$

For GPU-based algorithms

$$Speedup = \frac{\textit{the running time of the CPU version of the algorithm}}{\textit{the running time of the GPU version of the algorithm}}$$

Amdahl's law

- Amdahl's law states that the speedup achieved through parallelization of a program is limited by the percentage of its workload that is sequential.
- We can get no more than a maximum speedup equal to

$$\frac{1}{S + \frac{P}{N}}$$

- S is the percentage of the workload that remains sequential, P is the percentage of the workload that can be made parallel, N is the number of processors.

GPU computing in discrete optimization

- A literature survey in Schulz et al. (2013).
- About 100 publications on GPU computing in discrete optimization.
- Published during 2002-2012.
- Most papers discuss GPU implementation of metaheuristics.

Discrete optimization problems studied

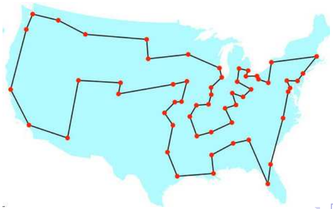
- Shortest path problem
- Traveling salesman problem
- Vehicle routing problem
- Task matching
- Flowshop scheduling
- Knapsack problem
- Quadratic assignment problem
- Graph coloring
- ...

Algorithms implemented on GPU

- Ant colony optimization (23)
- Particle swarm optimization (18)
- Population based (like genetic algorithm) (41)
- Local search (6)
- Tabu search (3)
- Simulated annealing (1)
- ...

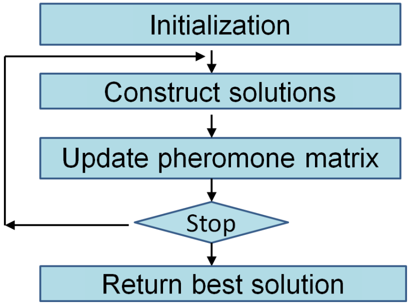
Traveling salesman problem (TSP)

- Next, three GPU based parallel metaheuristics for TSP.
- Given a list of cities and the distances between each pair of cities.
- Find the shortest tour that visits each city exactly once and returns to the origin city.



Parallel ant colony optimization on GPU

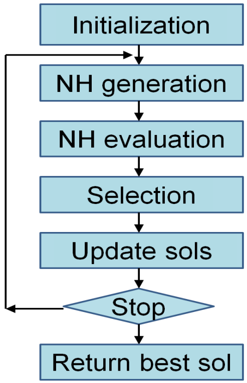
Ant colony optimization is based on the cooperation of a set of ants.



Parallel ant colony optimization on GPU

- Commonly put the tour construction on the GPU.
- Most algorithms put pheromone update on the GPU.
- A thread is assigned to compute the full tour of one ant (one-ant-per-thread).
- A thread computes only part of the tour and a whole block is assigned per ant (one-ant-per-block).
- One-ant-per-block approach seems to be superior to one-ant-per-thread.
- Speedup up to 29 for TSP (Schulz et al., 2013).

Parallel LS based metaheuristic on GPU



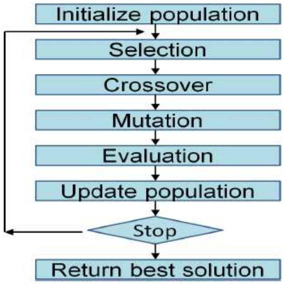
- Luong (2012) presents a template for LS based metaheuristics.
- CPU controls the whole sequential part of the algorithm.
- GPU does tasks like neighborhood evaluation.
- Suitable for best improvement.
- A tabu search using 2-opt is implemented for TSP.

Parallel LS based metaheuristic on GPU

- To put neighborhood generation and evaluation on GPU can reduce data transfer.
- For parallelism control, the total number of threads and the number of threads per block are dynamically selected.
- An approach to map each neighbor to each GPU thread.
- Global memory accesses are coalesced.
- Use texture memory to store problem inputs and solution representations.
- Speedup up to 19.9 for TSP.

Parallel genetic algorithm on GPU

Based on the evolvement of a population of solutions:



Parallel genetic algorithm on GPU

- CUDA-based genetic algorithm proposed by Chen et al. (2011).
- GPU does selection, crossover, mutation and evaluation.
- For parallelism control, one individual to one thread.
- Use shared memory to store part of the population.
- Speedup up to 1.7 for TSP.

Conclusions

- Easily accessible parallel resources.
- Well-suited to address problems that can be expressed as data-parallel computations.
- May achieve significant speedup.
- Discrete optimization on GPUs has become an interesting research field, still at its early stage.
- Hardware dependent, not easily portable.

References

Brodtkorb, A.R., Hagen, T.R., and Særa, M.L.(2013). GPU programming strategies and trends in GPU computing. *Journal of Parallel and Distributed Computing*, 73:4-13.
Crainic, T. G., and Toulouse, M.(2010). Parallel metaheuristics. In Gendreau, M. and Potvin, J.-Y., editors, Handbook of Metaheuristics, pages 497Ű541, New York. Springer.
Kirk, D., and Hwu, W.(2010) Programming Massively Parallel Processors: A Hands-on Approach, Morgan Kaufmann Publishers, 2010.
Laporte, G.(2007). What you should know about the vehicle routing problem. *Naval Research Logistics*, 54:811Ű819.
Leont'ev, V. K.(2007). Discrete optimization. *Computational Mathematics and Mathematical Physics*,47:328-340.
Luong, T.V.(2012). Parallel metaheuristics on GPU. PhD thesis, Lille 1 University - Science and Technology.
Martinez, G., Gardner, M., and Feng, W., CU2CL: A CUDA-to-OpenCL Translator for Multi- and Many-Core Architectures, in Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on, dec. 2011, pp. 300 Ű307.
Schulz, C., Hasle, G., Brodtkorb, A. R., and Hagen, T. R. (2013. GPU Computing in Discrete Optimization Part II: Survey Focused on Routing Problems. Submitted.
Chen, S., Davis, S., Jiang, H., Novobilski, A.(2011). CUDA-Based Genetic Algorithm on Traveling Salesman Problem. In: R. Lee (ed.) Computer and Information Science 2011, Studies in Computational Intelligence, vol. 364, pp. 241-252. Springer Berlin Heidelberg.

Thank you for your attention!