AUTOMATED MISSION PLANNING FOR MOBILE ROBOTS USING EMBEDDED GPU

SHIPOVALOV, R. & PRYANICHNIKOV, V.

Abstract: *Mission planning is a fundamental problem in mobile robotics. Domain-independent planners and the PDDL language provide a standard and flexible framework for solving it. However, such planners have high requirements for computing resources which are at a premium on mobile platforms; also, they are single-threaded CPU-based and hence do not scale well. GPGPU-enabled onboard computers such as NVIDIA Jetson family boast high TFLOPs and energy efficiency, but no automated planners described in scientific literature are able to take full advantage of them. This paper describes a new automated planner running entirely on the GPU and its application for service mobile robot mission planning.*

Key words: *automated planning, PDDL, GPU, GPGPU, mobile robot*

b

Authors´ data: Shipovalov, E[gor]*; Ph D., Prof. **Pryanichnikov**, V[alentin]**, ** Keldysh Institute of Applied Mathematics, Miusskaya sq., 4, Moscow, 125047, Russia, egorsh@gmail.com, v.e.pr@yandex.ru

This Publication has to be referred as: Shipovalov, E[gor] & Pryanichnikov, V[alentin] (2019). Automated Mission Planning for Mobile Robots Using Embedded Gpu, Chapter 30 in DAAAM International Scientific Book 2019, pp.351-358, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-24-2, ISSN 1726-9687, Vienna, Austria
DOI: 10.2507/daaam.scibook.2019.30

1. Introduction

Domain-independent planning is a process of automatic derivation of a plan (a sequence of actions) from a task formulated in a generic high-level language. The dominant language for automated planning today is PDDL (Bonet and Geffner, 2001). The origins of automated planning lie in robotics (Fikes & Nilsson) which remains its key application (Ai-Chang et al., 2004) (Barreiro et al., 2012) The solution described in this paper implements *classical planning* – a branch of planning characterised by complete determinism, where the state of the system can only be affected by actions described in the task, and their outcomes are always predictable.

Due to the state space explosion, planning problems are often intractable in the worst case, and when they are solvable, their resource requirements are massive and tend to grow exponentially. Multi-core high-performance platforms such as modern GPUs are generally suitable for solving problems of this type. However, due to difficulties of parallelising inherently sequential algorithms such as heuristic search, GPU use for automated planning has only begun recently and remains very limited (Horie and Fukunaga, 2017) GPUs either accelerate specific parts of a CPU-based planner (Sulewski, 2012) or implement niche search methods with limited applicability(Kuroiwa and Fukunaga, 2018). No publications appeared so far proposing an onboard mobile GPU implementation.

The described solution for service robot mission planning in indoor environments includes CUDA-based parallel planner and a PDDL task formulation engine. The planner handles STRIPS PDDL extended with negative preconditions, equality predicate and action costs. PDDL is generated dynamically based on text and/or SVG environment specification. Benchmarking on a representative range of GPU devices against an equivalent CPU implementation showed significant improvement in absolute performance as well as in performance per Watt.

2. Background and related work

The objective of *automated planning* is to find, without human intervention, a sequence of actions bringing the system from the initial state to some other state, satisfying the goal criteria. In *domain-independent planning*, the problem field (called a *domain*) and the problem itself are described a generic language that makes no assumptions about its application. A wide variety of tasks, for example, solving a chess etude, laying out a car route, making a part on a CNC machine, allocating financial instruments for maximum profitability, can be, under certain conditions, solved with domain-independent automated planning techniques.

Problems for domain-independent planning can be described using different models with varying complexity. Some more complicated models take real-world factors such as time, uncertainty or spontaniety into account. However, the most practically important are so-called *classical* models which are chatracterised by complete determinism, as these are most amenable to solving with current hardware and algorithms.

The first automated planners for classical models were proposed in the 1970's to control Shakey mobile robot (Fikes and Nilsson, 1972). Exponential complexity of planning initially confined automated planners to lab use, but in 1990's – 2000's, algorithmic and hardware improvements led to more practical applications, and their number continues to grow. However, the stagnation of CPU performance which stems from the fundamental limitations of silicon-based hardware, calls for new, scalable solutions, if significant advances are to be made.

In this paper we describe such a solution and its application. An indoor service mobile robot is paired with a highly parallel automated planner, able to scale throughout the entire range of NVIDIA CUDA devices: from low-power Jetson family SBC's targeted at mobile robots, to datacenter-based HPC clusters of Tesla V100 cards. Our tests show that this soultution outperforms an industry-standard CPU-based planner by a significant factor in both absolute performance, as well as in performance per watt. Another unique feature of the solution which is particularly important for resource-constrained mobile platforms, is that it performs the time-consuming search phase of planning on the GPU only, freeing up the CPU cores to handle time-critical tasks.

The use of GPU in automated planning was first described in a hybrid CPU-GPU system(Sulewski et al., 2011). Because it relies on the CPU during the search phase, its efficiency is severely limited by the overhead of moving data to and from GPU and re-executing the kernel at every state expansion. This is avoided in (Zhou and Zeng, 2015) by implementing parallel A* search entirely on the GPU. However, the level of parallelism in this solution is limited by the maximum number of threads per block (1024 on current hardware). An iterative deepening GPU-based variant of A* (IDA*) (Korf, 1985) was published in (Horie and Fukunaga, 2017). The downside of the this approach is the multiple re-expansion of the same nodes inherent in IDA* which is exacerbated by computational expensiveness of planning heuristics. Considering the shortcomings of the aforementioned approaches, it is fair to conclude that no scalable GPU implementation of best-first heuristic search suitable for use in planning has been described in scientific literature. For a detailed review of parallel A* implementations, see (Fukunaga et al., 2017).

3. CUDA as a platform for mobile robotics and artificial intelligence

CUDA has become the dominant platform in GPGPU and affordable supercomputing research. Its scalability is also unprecedented: CUDA applications can run without recompilation on 15-Watt mobile platforms as well as multi-KWatt data center clusers such as NVIDIA DGX-2 cluster with 16 GPUs and 80 000 cores.

This, however, comes at a cost: *the lock-step parallelism* in CUDA is, in most ways, more limiting than parallel CPU-based systems. CUDA *symmetric multiuprocessors (SM's)* group threads into bunches of 32 called *warps*, sharing the same instruction pointer. Such threads execute concurrently as long as the instruction pointer points at the same address. When a conditional operator evaluates differently in different threads and they diverge, they must take turns, so overall performance degrades.

To minimise such occurences, called *branch divergence*, special care needs to be taken at the algorithm design and implementation levels. In the described solution, most algorithms are designed with branch divergence minimisation in mind.

Another limitation of CUDA was, until recently, the lack of means of runtime collaboration between concurrently running instances of programs (CUDA blocks). Global memory coherency and synchronisation barriers became available with CUDA 9.0 and the Pascal architecture, of which our solution makes use.

	Jetson TX2	**GeForce 1050Ti**	**Tesla V100**
Arhitecture	Pascal	Pascal	Volta
Number of SMs	2	6	80
Memory type	LPDDR4	DDR5	HBM2
Termal Design Power	15	75	300

Table 1: Capability range of CUDA-enabled devices

4. Architecture of the solution

The planner consists of four main modules: shell, PDDL parser, translator and the search engine. Let us consider them in detail. The shell orchestrates internal modules and provides APIs for client software, task evaluation and functional testing. It is implemeted in C++ and build as a single executable. Two APIs are offered: batch command-line API supporting standard input and output sterams, and client-server HTTP-based API. When the planning completes, in addition to the found plan, shell outputs a statistical report, detailing task's key parameters and the volume of resources used.

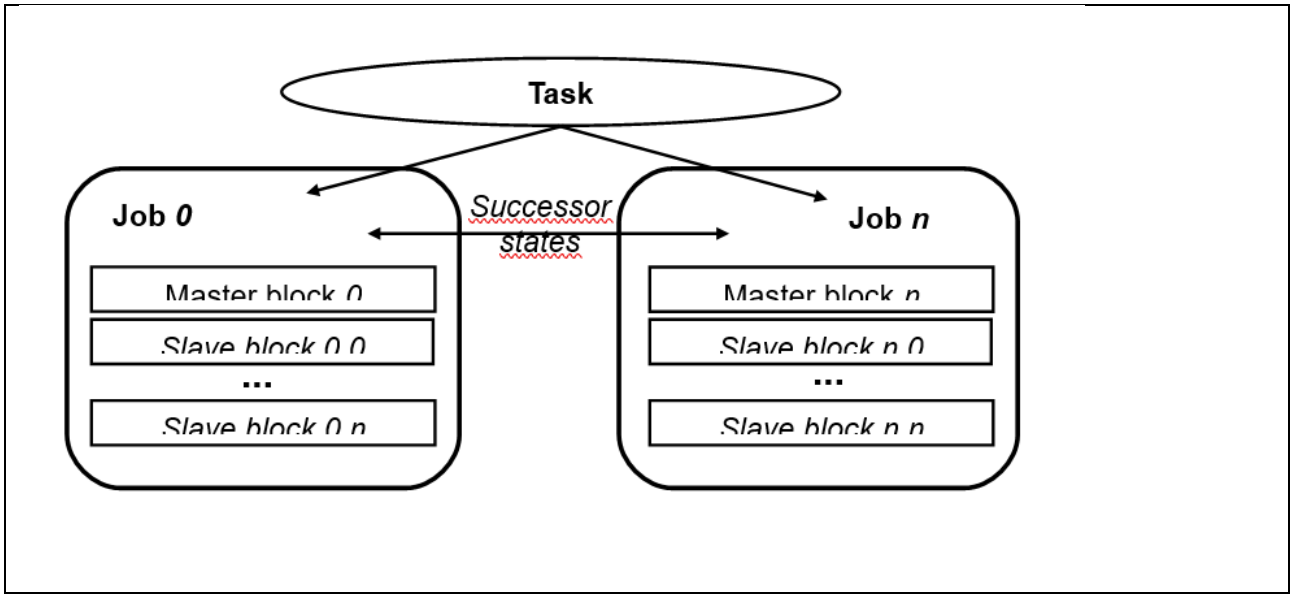
The syntax analyser check the correctness of PDDL domain and problem formulations. It then builds compact internal representaitons for these based on the abstract syntax tree. In contrast with the most automated planners, no compiler tools, such as Lex, Yacc or Bison are used. The Lisp-like syntax of PDDL allows to build a simple and fast recursive descent parser, with an added benefit of more descriptive error messages. The translator convers the intermediate task representation produced by the syntax analyser into a so-called *ground task* that has the data necessary for heuristic computation as well as state space search. It is a relativelty compact data structure that remains unchanged throughout the planning process. There are planning algorithms that operate on ungrounded, with so-called *lifted* tasks, but these are much slower and for this reason bear little practical significance.

Before grounding, the translator performs substitutions necessary to support extensions to the classical planning model: negative preconditions, the equality predicate, and a few others which make task definitions more compact without changing their mathematical properties.

Once the grounding is complete, the translator detects and drops invariant literals: ones, the truth values of which cannot change under this task's specification. For example, the property of being visible from one another for a pair of terrain points in a navigation task. This makes task more compact and the search faster. Then the set of literals is sorted to speed up comparisons, and the task is ready for the search phase.

The search is performed in one of three modes: A*, EHC (enforced hill climbing) (Hoffmann, 2001) , and IDA* (iterative deepening A*). The first can gurantee optimality of plans; the second generally finds the solution quicker, albeit not an optimal one; the third uses the least memory which can be useful on resource-constrained mobile platforms. The search mode is specified explicitly based on the task and plan requirements. Once a goal state is found during the search, a plan is formed by traversing state ancestors and sent to the API client.

Parallelisation of the search is achieved by assigning two operators to each CUDA thread and dividing the search space using hash-based a distrubution scheme (Kishimoto et al., 2013). Each state space subdivision is handled by a separate search process (job). As jobs produce successors during search node expansion, they distribute these among themselves based on their hash values. A job may also contain multiple CUDA blocks, which work in a master-slave pattern. The slave blocks do not search, but only compute heuristic estimates for the states produced by the master.



Drawing 1: Search parallelisation

5. Robot environment description

The planning software requires tasks to be formulated in PDDL language whereas robot's environment and mission objectives are described using application-specific data structures. This gap is bridged in the following way. The indoor roadmap traversable by the robot is defined as a directed graph in a text configuration file. Each vertex of this graph corresponds to a named landmark identifiable by robot's onboard camera. This allows the robot to navigate any route specified as a list of landmarks as long as the bearing between two neighboring landmarks is known. The bearing and,

optionally, distance values between pairs of connected vertices are included in the roadmap specification as will be shown below. Knowing the distance value allows the robot to travel faster than accurate landmark recognition requires, and slow down when it is approaching the next landmark. In practice, distance and bearing are either measured on location using a laser range finder and a laser angle, or read and computed from an SVG format file. Here's a minimal example of roadmap specification with three locations (the “leg” statement defines bearing and distance):

```
location start 679 1399
location turn 680 1272
location end 1299 1282
leg start turn 1 127
leg turn start 181 127
leg turn end 90 619
leg end turn 270 619
```

Table 2: Roadmap definition example

In this roadmap graph there are two edges for each landmark pair. Should there be, for example, a stairway on which the robot can travel down but not up, this would be supported by having just one edge. Alternatively, we have developed a converter that can read a roadmap specification from an SVG file, compute bearings and distances automatically and produce a route definition file. This is convenient when map of the area is available, so a roadmap can be specified by drawing vector paths over it with a graphics package such as Inkscape.

The objects that the robot can transport between locations vary in size, shape and weight. The space on the robot's cargo platform is limited as well, and so is its payload. There are also natural physical limitations on how transported objects can be stacked on the platform. These requirements are captured in a text configuration file where each object is configured with the following parameters: name, length, width, height,weight, max. load. The last parameter tells how much load can be placed on the object, with 0 being a valid value (nothing can be placed). Here's an example of an items configuration (note how 5KG of load can be placed on a book, but nothing of other items):

```
item water-bottle       110 110 390  1000     0
item english-dictionary 300 250  30  2000 50000
item cordless-drill     262  85  85  2000     0
```

Table 3: Transported items description example

Once the environment has been described, it can be compiled into PDDL. Initial and desired locations of items (the latter constitute the mission goal) are specified in a text file and combined with the above data to produce problem definition. Custom PDDL domain is generated for each environment, but can be reused for different missions. The generation is implemented with generic templates adhering to Mustache specification which are processed using environment and mission definition. The

PDDL definition is then passed over to the planner for solving. Once a plan is ready, it is interpreted as a list of commands for the robot to execute. The PDDL actions directly correspond to commands supported by AMUR-307 mobile robot: move, pick-up, put-down, load, unload, stack, unstack.

6. Results

The solution was tested on the following Nvidia GPU systems: Jetson TX2, , GeForce 1050Ti, Tesla V100 (via Amazon Web Services). The results were compared to single-threaded the reference CPU implementation of the same algorithms, implemented in C++ on Intel i7-6800K CPU clocked at 3.4 Ghz with DDR4 memory. The reference implementation generates state spaces identical to single-job GPU imlementation and is also used for automated verification during development. This ensures identical behaviour of the CPU and GPU algorithms and hence the validity of comparison. The comparative performance for a batch of three planning tasks of varying difficulty (easy, medium, small) was as follows:

	Jetson TX2	**1050Ti**	**Tesla V100**
Architecture	Pascal	Pascal	Volta
Number of Multiprocessors	2	6	80
Number of cores	256	768	5120
Memory type	LPDDR4	DDR5	HBM2
Thermal Design Power*, Watt	15	75	300
Number of master jobs	4	5	20
Number of slave jobs per master	0	1	6
Total jobs	4	10	140
Time to solve the „hard“ task, sec.	88	30	18
Time to solve the „medium“ task, sec.	21	12	6
Time to solve the „easy“ task, sec.	16	2	1
Average processed states per second	97601	161 323	1 412 595
Speed-up, times relative to CPU version	5	8	94
Performance per increase, rel. to CPU	47	16	50

Number of slave jobs per master was set explicitly as an educated guess based on task size and the number of cores. In general, the more cores the GPU has, the higher should be the share of slave blocks in total number of blocks. Master-to-slave block communication volume is significantly lower than master-to-master so we can get more CUDA blocks to run before communication overhead starts to dominate the run time, making futher increases in the number of blocks worthless.

7. Conclusions and future work

We described a new GPU-based domain-independent planner with integration middleware deployed on a service mobile robot. Tests demonstrated significant advances in scalability and performance per Watt relative to CPU-based reference implementation. A novel method of translating mobile robot mission planning problems into PDDL representation was also presented. These make a significant step towards efficient and accessible autonomous service robots. Future research will concentrate on adding support for more PDDL features, including temporal planning.

8. References

Ai-Chang, M., Bresina, J., Charest, L., Chase, A., Hsu, J.C., Jonsson, A., Kanefsky, B., Morris, P., Rajan, K., Yglesias, J., others, 2004. Mapgen: mixed-initiative planning and scheduling for the mars exploration rover mission. Intelligent Systems, IEEE 19, 8–12.

Barreiro, J., Boyce, M., Do, M., Frank, J., Iatauro, M., Kichkaylo, T., Morris, P., Ong, J., Remolina, E., Smith, T., others, 2012. EUROPA: A platform for AI planning, scheduling, constraint programming, and optimization, in: Proceedings of the 22nd International Conference on Automated Planning & Scheduling (ICAPS-12)–The 4th International Competition on Knowledge Engineering for Planning and Scheduling.

Bonet, B., Geffner, H., 2001. Planning as heuristic search. Artificial Intelligence 129, 5–33.

Fikes, R.E., Nilsson, N.J., 1972. STRIPS: A new approach to the application of theorem proving to problem solving. Artificial intelligence 2, 189–208.

Fukunaga, A., Botea, A., Jinnai, Y., Kishimoto, A., 2017. A Survey of Parallel A*. arXiv:1708.05296 [cs].

Hoffmann, J., 2001. FF: The fast-forward planning system. AI magazine 22, 57.

Horie, S., Fukunaga, A.S., 2017. Block-parallel IDA* for GPUs, in: Proceedings of the Tenth International Symposium on Combinatorial Search, Edited by Alex Fukunaga and Akihiro Kishimoto. pp. 16–17.

Kishimoto, A., Fukunaga, A., Botea, A., 2013. Evaluation of a simple, scalable, parallel best-first search strategy. Artificial Intelligence 195, 222–248.

Korf, R.E., 1985. Depth-first iterative-deepening: An optimal admissible tree search. Artificial intelligence 27, 97–109.

Kuroiwa, R., Fukunaga, A., 2018. Batch Random Walk for GPU-Based Classical Planning., in: ICAPS. pp. 155–160.

Sulewski, D., 2012. Large scale parallel state space search utilizing graphics processing units and solid state disks.

Sulewski, D., Edelkamp, S., Kissmann, P., 2011. Exploiting the Computational Power of the Graphics Card: Optimal State Space Planning on the GPU.

Zhou, Y., Zeng, J., 2015. Massively Parallel A* Search on a GPU., in: AAAI. pp. 1248–1255.