Real-Time Power-Efficient Integration of Multi-Sensor Occupancy Grid on Many-Core

Tiana Rakotovao[1,2], Julien Mottin[1], Diego Puschini[1], Christian Laugier[2]

***Abstract*— Safe Autonomous Vehicles (AVs) will emerge when comprehensive perception systems will be successfully integrated into vehicles. Advanced perception algorithms, estimating the position and speed of every obstacle in the environment by using data fusion from multiple sensors, were developed for AV prototypes. Computational requirements of such application prevent their integration into AVs on current low-power embedded hardware. However, recent emerging many-core architectures offer opportunities to fulfill the automotive market constraints and efficiently support advanced perception applications. This paper, explores the integration of the occupancy grid multi-sensor fusion algorithm into low power many-core architectures. The parallel properties of this function are used to achieve real-time performance at low-power consumption. The proposed implementation achieves an execution time of $6.26ms$, $6\times$ faster than typical sensor output rates and $9\times$ faster than previous embedded prototypes.**

I. Introduction

During the last decades, a high interest on the development of Autonomous Vehicles (AVs) have been noticed in the field of research and industries [1], [2], [3]. Autonomous navigation is ensured by several critical subsystems fulfilling different required functionalities (obstacle detection, obstacle avoidance, adaptive cruise control, etc). A major subsystem on which rely the other components of AVs is the **environment perception module** which models the traffic scenario in a mathematical representation that a computer can understand and interpret to make further decision. Future perception module will require not only to detect the location of obstacles, but also predict and anticipate their movements, even when temporarily occluded by another object or in blind spots.

One of the common computational frameworks for building perception modules for AVs is the Occupancy Grid (OG). An OG maps the external environment of AVs into a grid composed of several cells as shown in Figure 1. Several sensors (lidar, camera, radar, sonar, etc.) are mounted on board in order to observe the surrounding of the vehicle. Each sensor has a probabilistic model, called Sensor Model (SM) that reflects its observation and measurement errors (noise, failures, physical limitation, etc) [4]. Based on sensor models, an OG algorithm performs a fusion of all the observations of on board sensors in order to computes the probability that each grid cell is *occupied* by an obstacle (see figure 1(b)).

Several perception algorithms based on OGs were developed in the literature. The notion of OG was first introduced in [5]. The author also presents the bayesian filter which is a probabilistic mechanism that allows to update OG computed at time t_{n-1} with OG produced by sensor observations at time t_n. In [6], [7], the Bayesian Occupancy Filter (BOF) and the Hybrid Sampling Bayesian Occupancy Filter (HS-BOF) utilize OGs for monitoring dynamic environments and tracking mobile objects. Besides perception, OGs are also used for other functionalities such as road boundary detection [1], pedestrian detection and tracking [8], simultaneous localization and mapping (SLAM) [4].

OGs constitute a dominant paradigm for environment perception because of their ability to take into account measurement uncertainties and errors, and their possibility to efficiency support multiple sensors to get a more accurate estimation of obstacle position. However, OG algorithms suffer from the amount of computational operations they require to map a large surrounding area. Increasing the size of the grid leads to adding more cells, unless augmenting the cell size which prompts to a loss of precision. A high number of cells induces high amount of computation. OG algorithms are then often implemented on powerful highly parallel hardware like Graphics Processing Units (GPUs) [1], [9] to perform computations in real-time, ensuring that OGs are produced at the same rate as sensors measurements. However, GPU platforms do not meet AVs requirements in terms of power consumption, size and cost. To overcome this limitation, a first implementation of OG-based perception algorithm on an embedded many-core is presented in [10]. While the implementation is power-efficient, OGs are still not produced in real-time.

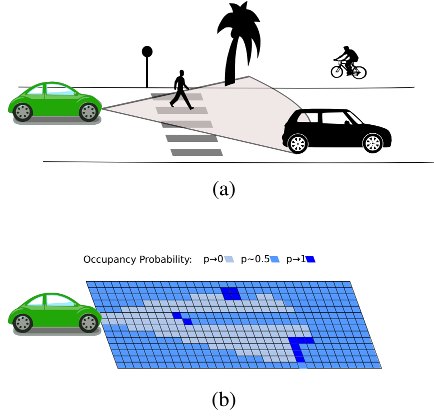
In this paper, we present a real-time and power-efficient computation scheme for occupancy grids on many-core, by fusing observations from multiple lidar sensors. This scheme relies on a method for implementing OG computation, based on the fact that lidar sensors are composed by several beams that can be processed independently. Experiments show that with the proposed approach, OGs can be computed within less than $7ms$ on an embedded many-core consuming at most $1W$ of electrical power.

The paper is organized as follows. Section II presents an overview of the OG algorithm and its main steps. Its integration on many-core is detailed in Section III. After that, experimental results are presented in Section IV. Finally, Section V concludes the paper.

II. Occupancy Grid Algorithm

Occupancy Grid (OG) computation is based on probabilistic calculus and Bayes' rule on conditional probability. The

[1]CEA-LETI MINATEC Campus, 17 rue des Martyrs, 38000 Grenoble
[2]INRIA Grenoble Rhône-Alpes, 655 avenue de l'Europe, 38334 Saint Ismier cedex



(a)

(b)

Fig. 1. Occupancy Grid Principle. (a) Scenario and sensor. (b) Occupancy Grid modeling the scene.

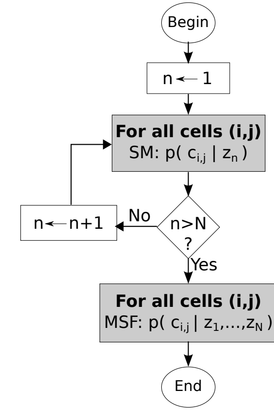


Fig. 2. Overview of Occupancy Grid Algorithm

main steps for building OGs from multi-lidar observation are detailed in this section. No notion of grid filtering is presented, that is, the occupancy grid OG_t at time t is built only from currently available sensor measurements z_t. Previous occupancy grid OG_k with $k < t$ are not taken into account.

A. Algorithm Overview

As shown in Figure 1, occupancy grid maps the environment into a spatial grid with regular cells. The probability of each cell for being *occupied* is then computed according to available sensor measurements.

In the context of AV perception, the occupancy grid is mainly used for updating a local map of a restricted region in the surrounding of the ego-vehicle. For instance, in [6], [7], a map of $30m \times 50m$ in the front of the vehicle is monitored. This approach is different of a Simultaneous Localization and Mapping (SLAM) problem where the vehicle simultaneously builds a global map of the environment (eg: a complete map of a city) and simultaneously tries to localize itself relative to the map [4].

In the case of the AV perception, let $\mathcal{G}$ be the virtual spatial grid placed in the front of the ego-vehicle, formally defined by:

$$\mathcal{G} = \{(i,j)\ ,\ i \in \{1, \cdots, I\}\ ,\ j \in \{1, \cdots, J\}\} \tag{1}$$

The couple (i,j) denotes a cell in the grid, I and J design the number of rows and columns of the grid. Let c be a function which makes the correspondence of a cell (i,j) to a binary state: occupied o or free $\overline{o}$. Equation 2 gives a formal definition of c.

$$\begin{array}{cccc} c & : & \mathcal{G} & \mapsto & \{o, \overline{o}\} \\ & & (i,j) & \mapsto & c(i,j) \end{array} \tag{2}$$

For the sake of simplification , let us define:

$$\begin{array}{c} c_{i,j} \equiv (c(i,j) = o) \\ \overline{c}_{i,j} \equiv (c(i,j) = \overline{o}) \end{array}$$

The occupancy grid OG is then defined by the following function:

$$\begin{array}{cccc} OG & : & \mathcal{G} & \mapsto & [0,1] \\ & & (i,j) & \mapsto & p(c_{i,j}|z_1, \cdots, z_N) \end{array} \tag{3}$$

where $z_1, \cdots, z_N$ denote the measurements from sensor 1 to sensor N.

For simplifying the computation of the probability distribution in equation 3, the following hypothesis are considered:

1) the state of a cell is independent of the state of any neighboring cell,
2) sensor measurements are independent one from another, that is, a measurement from one sensor does not impact the observation from another sensor.

The two hypothesis allow to simplify the distribution $p(c_{i,j}|z_1, \cdots, z_N)$ of equation 3 as a function of distributions $p(c_{i,j}|z_n)$, $n \in \{1, \cdots, N\}$. Given a unique sensor n, $p(c_{i,j}|z_n)$ describes the state of the cell (i,j) according to measurement from that sensor. This distribution is called *Sensor Model (SM)* of sensor n. It models the measurement errors, the measurement noise and the physical limitation of the sensor into a probabilistic distribution. Once all SMs from all sensor observations are computed, they are fused into a unique distribution per cell $p(c_{i,j}|z_1, \cdots, z_N)$ which expresses the occupancy probability of each cell. This last process is called *Multi-Sensor Fusion (MSF)*.

Figure 2 shows a general overview of occupancy grid algorithms. It is divided into two parts. First, the SMs $p(c_{i,j}|z_n)$, $n \in \{1, \cdots, N\}$ regarding all sensors are computed on each cell. Then, the SMs are fused through the MSF to get the final occupancy probability $p(c_{i,j}|z_1, \cdots, z_N)$ per cell.

B. Sensor Model (SM)

Sensors constitute a fundamental element for perception systems. They provide physical measurements on the real scenario on the surrounding of the vehicle. However, sensor measurements are not exact. Measurement incertitude should be taken into account when modeling the observation into a computational map such as OGs. SM allows to express measurement incertitude of a sensor into occupancy probability.

The present work focus on lidar sensors. They are composed of several beams that operate independently. Formally, a lidar measurement is actually a set $z_n = \{z_n^1, \cdots, z_n^B\}$ of B independent measurements, where B is the number of beam within the sensor. For simplifying the notation in the following paragraphs, we denote by z the measurement from a single beam.

Lidar sensors emit beams and record their time-of-flight. For each beam, the time-of-flight serves to compute the distance z, also called *range*, to the obstacle hit by the beam. The range is assumed to be close to the true position z^* of the obstacle [11], [1], but uncertainty still persists due to measurement noises, the physical nature of the obstacle that might badly reflect the light of the beam, the incidence angle of the beam on the surface of the obstacle or any other uncertainty.

A beam is emitted in a linear fashion towards a given direction in the physical world. Its SM is presented in equation 4.

$$p(o_d|z) = \begin{cases} max(\overline{p}(d,z), g(d,z)) & \text{if } d \leq z \\ max(0.5, g(d,z)) & \text{if } z < d \end{cases} \quad (4)$$

where

$$g(d,z) = \lambda(d) exp\Big[-\frac{1}{2} \frac{(d-z)^2}{\sigma_d^2} \Big]$$

$g(d,z)$ is a Gaussian which describes the range measurement uncertainties. The SM $p(o_d|z)$ evaluates the probability that an obstacle exists at a distance d from the source of the beam [12]. Figure 3 gives an idea of the curve of equation 4 as a function of distance d from the origin of the beam, in the case where the first hit obstacle is located at $z = 6m$ from the source of the beam.

Figure 3 shows that the probability that there is actually no obstacle before the range is not null. This probability is called as the *free distribution*. It is evaluated by $\overline{p}(d,z) = p(\overline{o}_d|z)$, $d < z$ in equation 4. The free distribution can be defined as a function of d to show that the beam hardly miss nearer obstacles than further.

In the Gaussian function $g(d,z)$, $\lambda(d)$ represents the amplitude. It evaluates the confidence given to the Gaussian regarding the distance d from the source of the beam. One can also notice that the standard deviation σ_d of the Gaussian is a function of d. The sensor might be more precise at near distances while less accurate at far distances. Finally, the second line of equation 4 shows that the beam does not provide any information about the occupancy behind the hit distance. The SM returns 0.5 from a certain distance behind the range (see figure 3) which means that it does not give any idea of the existence or not of any obstacles from that distance.

C. Multi-Sensor Fusion (MSF)

The MSF equation is proposed by [5], albeit several approaches exist in the literature [4], [12]. MSF fuses two occupancy probabilities on the same spatial location, regarding measurements from two sensors. The sensors are assumed to operate separately. In the case of a 2-dimensional environment, the formulation of MSF is given as follows:

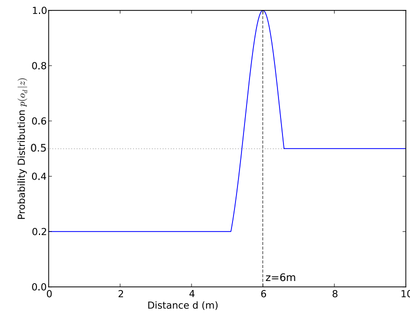


Fig. 3. Beam Sensor Model $p(o_d|z)$ for $z = 6m$, $\sigma = 0.5$

$$p(o_{x,y}|z_1, z_2) = \frac{p_1 p_2}{p_1 p_2 + (1-p_1)(1-p_2)} \quad (5)$$

where (x,y) are the coordinates of a spatial point, and $p_n = p(o_{x,y}|z_n)$ is the SM regarding sensor n.

Equations 4 and 5 are defined over continuous physical space. Occupancy grids are only a simple manner to discrete that space. Spatial coordinates or distance are transformed to cell location. The next section explains how to implement the occupancy grid algorithm on an embedded hardware based on many-core architecture.

III. Occupancy Grid Integration on Many-Core

Due to the independence hypothesis of cells and of sensor measurements, the *for* loops in occupancy grid algorithm on figure 2 can be implemented in a parallel fashion. In order to embed the algorithm into an autonomous vehicle, it has to be integrated into a low-power embedded system. Such hardware must offer enough computing power for processing occupancy grid computation in real-time: at least, occupancy grid output rate is equal to sensor measurement output rate. Many-core hardware accelerators are good candidates for satisfying such requirements [10]. Their main characteristics consist on a low power consumption while providing a relatively high peak computing performance [13].

A. Hardware Architecture

Figure 4 shows the generic template of a computing hardware with a many-core accelerator. It is divided into two main parts:

- the *CPU host* is generally a general purpose CPU dedicated for running an operating system and common sequential applications, and for communicating to the external world through different input/output interfaces (UART, Ethernet, GPIO, USB, etc.)
- the *many-core* is a hardware accelerator composed of dozens of computing cores called *Processing Elements* (PE). A PE generally disposes less advanced features than the host (control unit, cache management, instruction set, etc.). PEs can run programs in a Multiple Program Multiple Data (MPMD) fashion.

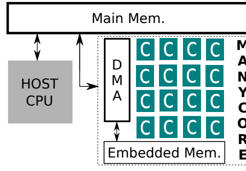


Fig. 4. Generic Architecture of Many-core

While each PE can have its own set of registers, the many-core also disposes an embedded on-chip fast memory, shared by all PEs. Due to the limit of the memory space available on the embedded memory (generally less than 1MByte), the PEs can also access to an external main memory which is shared by the many-core and the host. However, such access is slow and time consuming. DMA blocks are then used for accelerating data transfer between the two memory regions (see figure 4).

While each PE can execute its own program independently, some software and hardware features allow efficient communication, synchronization and data sharing between cores [13]. Consequently, many-core can be programmed to compute data parallel or task parallel problems. For a data parallel problem, the data is split into several sub-data that should match the number of available PEs. Each sub-data is then processed by one PE, independently of the other PEs or in synchronization with them if necessary. On the programming side, many-core can support well known parallel programming standard such as MCAPI, OpenCL [14], OpenMP [13], lightweight POSIX threads [15].

B. Computing Occupancy Grids on Many-core

The algorithm on figure 2 is divided into two main loops: the computation of sensor models for all cells, and the multi-sensor fusion per cell. These loops can be implemented in a parallel fashion separately. Both of them can be expressed as a data parallel problem where instead of having a single processor looping on all cells, each cell is computed by a specific core on a parallel architecture.

However, the definition of MSF in equation 5 allows to incrementally fuse SMs. That means, $p(c_{i,j}|z_1)$ and $p(c_{i,j}|z_2)$ are first computed. Then they are immediately fused to get $p(c_{i,j}|z_1, z_2)$. After that, $p(c_{i,j}|z_1, z_2)$ and $p(c_{i,j}|z_3)$ are fused to get $p(c_{i,j}|z_1, z_2, z_3)$ and so on. Such approach is detailed in the following paragraph.

1) Cell-by-Cell (CBC) Approach: Measurements from multiple lidar sensors are used for building occupancy grids. A lidar disposes several beams. Beams are emitted in several directions separated by a constant angular step. Lidar measurement z_n represents the set of ranges returned by beams. Denote by z_n^b the range measured by the b-th beam.

Figure 5(a) presents the cell-by-cell approach for implementing the occupancy grid computation on many-core. It resumes the algorithmic steps for only one cell. A PE selects first a cell (i, j) which occupancy probability is to be computed. After that, for a lidar sensor n, the ordinal number b of the first beam which direction passes through the cell is calculated. Then, the SM and MSF equations are applied to the cell given the range of the found beam. The cartesian distance between the origin of the lidar sensor and the center of the cell is used for computing the sensor model distribution. Next, the same set of operations are repeated for all lidar sensors.

The advantage of such implementation is that cells are processed independently and no synchronization is needed. However, the number of PEs on many-core is typically much lower than the number of cells. Thus, the computations of occupancy probabilities of several cells are aggregated and executed by one PE. If x number of PEs are used, the grid is then divided into x sub-grids and each PE processes all cells within one specific sub-grid.

However, despite the aggregation, the number of cells to be processed by one PE remains high. This is a drawback for an implementation on many-core because when the amount of memory occupied by the grid cannot fit into the embedded on-chip memories (and it is often the case in practice), the occupancy probabilities and other additional data have to be stored in the main memory. Because the access time to the main memory from PEs is slow, the latency is reduced by the use of DMA blocks and by applying more advanced programming techniques such as double buffering [10]. However, the latency is not completely removed despite of DMA and increases with the number of cells.

Moreover, by using equation 5 to fuse $p_1 = 0.5$ and $p_2 = p$, being p any probability $p \in [0, 1]$, we notice that:

$$p(o_{x,y}|z_1, z_2) = \frac{0.5 \cdot p}{0.5 \cdot p + 0.5 \cdot (1 - p)} = p \qquad (6)$$

Equation 4 and figure 3 show that behind a certain distance from the range returned by a beam, the output of the SM is 0.5. Thus, performing the MSF is not necessary anymore due to equation 6. It will not change the numerical value of the occupancy probability.

2) Beam-by-Beam (BBB) Approach: To overcome the drawbacks of CBC implementation of OG algorithm on many-core, another method called Beam-by-Beam (BBB) approach is explored. The main difference between the two approaches resides in the data granularity of the parallelization. For the BBB approach, the least data granularity is a beam, while it is a cell for the CBC method. This means that in an ideal case where the many-core disposes a big number of cores, a PE would process a cell for CBC, while it would process a beam for BBB.

Providing that the number of beams traversing a cell is not uniform for all cells, we define the field-of-view of a beam b as the area delimited by the bisector of beams $b-1$ and b, and the bisector of beams b and $b+1$ (see figure 6). Consequently, the field-of-views of two beams of the same lidar sensor never overlap each other. To seek the beam which direction traverses a given cell, we assume that the cell is passed only by one beam per sensor: the beam which field-of-view contains the center of the cell.

Figure 5(b) depicts the main steps of computation for one beam. N is the number of lidar sensors. We assume that the

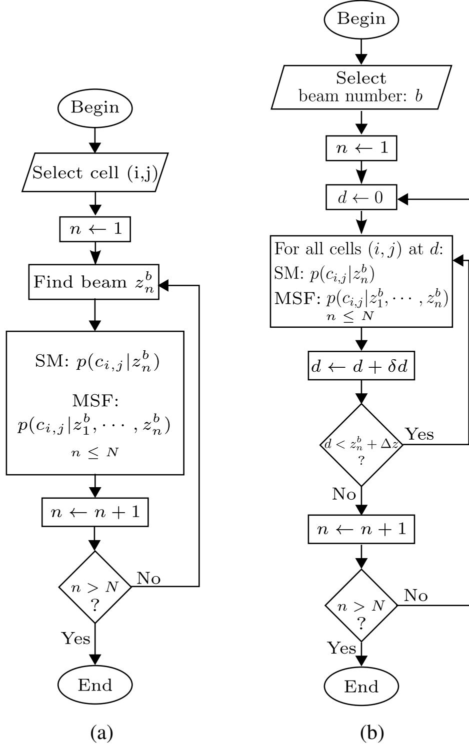


Fig. 5. Occupancy Grid Implementations: (a) cell-by-cell. (b) beam-by-beam.

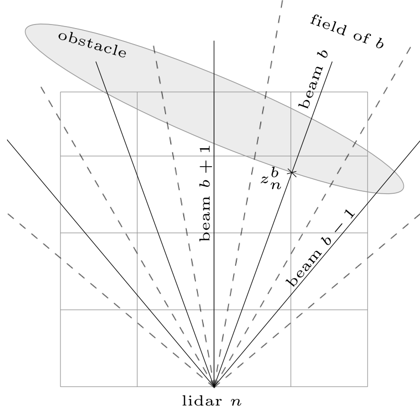


Fig. 6. Field of view of a beam

number B of beams per sensor is constant. First, each PE begins computations by selecting an ordinal number b of a beam among all B beams. Then, for a sensor n, the b-th beam is selected and a distance d is added from the origin to the direction of the beam. Next, the PE finds all cells (i, j) within the field-of-view of the beam, located at distance d. As shown on figure 6, the number of cells within the field-of-view increases with d. The occupancy probabilities of these cells are then updated through the SM and the MSF equations.

After that, a ray-casting in the direction of the beam is performed. The distance d is increased by δd which is chosen so that the next value of d will be located at cells in the direction of the beam, but which occupancy probabilities are not yet updated. The calculation of a beam stops as soon as the value of d becomes more than a certain distance Δz behind the range z_n^b returned by the beam. The PE processes then the next sensor and repeat the same steps above for the b-th beam of that sensor.

Because the number of PEs on many-core is less than the number of beams in a lidar sensor, the computation is still aggregated. The beams in a lidar sensor are divided into sub-groups so that a PE processes one sub-group. The intuition behind the beam-by-beam approach is driven by the fact that the number of beam is much lower than the number of cells. Consequently, the number of cells in a sub-grid processed by one PE is much higher than the number of beams in a sub-group. Then, aggregating beams matches better the number of computing resources available on many-core. Moreover, the BBB approach also takes advantage of equation 6. The number of operations is significantly reduced by stopping computations once the beam casting surpasses a certain distance from the range returned by the beam.

The BBB implementation on many-core presents a drawback. The occupancy grid is stored in the main memory in a row-major order or a column-major order. Because accessing directly the main memory from a PE is slow, one can use DMA transfers for reducing latency. However, the ray-casting-like algorithm access the memory not in row or column order, but in the direction of beams. Consequently, PEs do not necessarily access contiguous memory when performing ray-casting. This introduces a complication in DMA transfer management because DMA blocks outperforms better for contiguous block. A possible solution is to use 2-dimensional DMA transfer,where a small rectangular 2-d block of memory is simultaneously transfered between the main memory and the embedded memory. However, such approach also introduces an additional overhead because not all cells within the 2-d block are traversed by the beam. A cell outside the field-of-view of beam b but traversed by beam $b+1$ might be included within the 2-d block transfered by the PE processing the beam b. Consequently, if beam b and beam $b+1$ are processed by two different PEs, further synchronization between PEs is needed.

IV. Experiences

A. Experimental Setup

For experimental purposes, data collected by two IBEO Lux lidar sensors placed in the front left and front right of an ego-vehicle are used for testing the implementation of CBC and BBB approaches on many-core. Each sensor provides four independent layers of up to 200 beams each. The angular resolution of layers is $0.5°$. Beams can reach ranges up to $200m$. A lidar sensor produces four scans (one per layer) within a period of $40ms$. More details on the experimental platform (lidars and vehicle) can be found in [9].

The hardware test architecture comprises an embedded host CPU composed of dual ARM Cortex A9 @800MHz, and a low-power many-core accelerator with 64 cores and 2MBytes of on-chip embedded memory [13]. The cores are

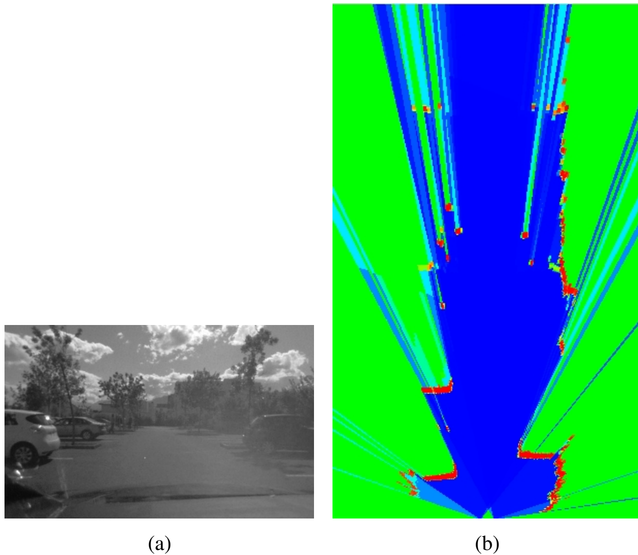

(a) (b)

Fig. 7. Experience on a parking: (a) scenario (Photo courtesy of INRIA.). (b) Occupancy grid.

grouped into 4 clusters of 16 cores. Each cluster features a dual channel DMA unit for transfers between main and embedded memory.

B. Execution Time and Power Consumption

Figure 7 shows an occupancy grid with the corresponding real scenario on a parking. Figure 7(a) is obtained from a camera mounted on the windshield of the ego-vehicle while figure 7(b) is produced by the execution of the beam-by-beam approach on the many-core hardware. A grid of $50m \times 30m$ with cells of $0.1m \times 0.1m$ is tested. The grid contains 150000 cells. Table I resumes the execution time and the power consumption for the implementations of the two approaches.

Concerning the execution time, with the BBB method, the many-core produces an occupancy grid with a period of $6.26ms$ which is about $6\times$ faster than the period of the lidar sensors. The occupancy grids are then computed in real-time compared to sensor output rates. Moreover, a remaining duration of $33ms$ can still be exploited to perform additional computations such as filters with still keeping the real-time constraint.

Approach	Execution Time(ms)	Power Consumption (W)
BBB	6.26	0.61
CBC	57.82	0.98

TABLE I
EXECUTION TIME AND POWER CONSUMPTION

In addition to being executed $9\times$ faster than CBC, the beam-by-beam approach consumes $1.6\times$ less electrical power. For a comparison with the state of the art, a grid with the same size but with cells of $0.2m \times 0.2m$ is also tested. The many-core performs the beam-by-beam approach at $3.5ms$. However, with the same parameters, an implementation of the cell-by-cell approach on a NVIDIA GPGPU GeForce GTX 480 lasts $8ms$ in average [9]. While the two experiments are not based on the same set of data, one can notice that the minimum recommended system power for the GPU is about 550W [16] however the maximum power consumption of the experimental many-core hardware is $1W$ [13].

V. CONCLUSION

This paper, has explored the integration of the Occupancy Grid (OG) Multi-Sensor Fusion (MSF) algorithm into low power many-core architectures. Two different approaches have been evaluated regarding parallel computing. The achieved execution time, $6.26ms$, were $6\times$ faster than used lidar output rate and $9\times$ faster than previous embedded prototypes, leaving enough free time for the other function required for AV. The power consumed by the many-core implementation remained less than $1W$, hundreds of times less consumption than a powerful GPGPU for a comparable execution speed. This implementation proves that real-time low-power multi-sensor fusion for occupancy grid is possible when many-core architectures are considered.

REFERENCES

[1] F. Homm *et al.*, “Efficient occupancy grid computation on the gpu with lidar and radar for road boundary detection,” 2010.

[2] A. Broggi *et al.*, “Extensive tests of autonomous driving technologies,” *Intelligent Transportation Systems, IEEE Transactions on*, 2013.

[3] M. Birdsall, “Google and ite : the road ahead for self-driving cars,” *ITE Journal (Institute of Transportation Engineers)*, 2014.

[4] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[5] A. Elfes, “Using occupancy grids for mobile robot perception and navigation,” *Computer*, 1989.

[6] C. Coué, C. Pradalier, C. Laugier, T. Fraichard, and P. Bessiere, “Bayesian Occupancy Filtering for Multitarget Tracking: an Automotive Application,” *International Journal of Robotics Research*, 2006.

[7] A. Negre, L. Rummelhard, and C. Laugier, “Hybrid sampling bayesian occupancy filter,” in *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*.

[8] S. Sato *et al.*, “Multilayer lidar-based pedestrian tracking in urban environments,” in *Intelligent Vehicles Symposium (IV), 2010 IEEE*, 2010.

[9] Q. Baig, M. Perrollaz, and C. Laugier, “A robust motion detection technique for dynamic environment monitoring: A framework for grid-based monitoring of the dynamic environment,” *Robotics Automation Magazine, IEEE*, 2014.

[10] T. A. Rakotovao, D. P. Puschini, J. Mottin, L. Rummelhard, A. Negre, and C. Laugier, “Intelligent vehicle perception: Toward the integration on embedded many-core,” in *Proceedings of PARMA-DITAM ’15*, 2015.

[11] K. Konolige, “Improved occupancy grids for map building,” *Autonomous Robots*, 1997.

[12] J. D. Adarve, M. Perrollaz, A. Makris, and C. Laugier, “Computing Occupancy Grids from Multiple Sensors using Linear Opinion Pools,” in *IEEE ICRA*, 2012.

[13] D. Melpignano *et al.*, “Platform 2012, a many-core computing accelerator for embedded socs: Performance evaluation of visual analytics applications,” in *Proceedings of the 49th Annual Design Automation Conference*, 2012.

[14] “Parallella,” www.parallella.org.

[15] B. de Dinechin *et al.*, “A clustered manycore processor architecture for embedded and accelerated applications,” in *HPEC, 2013 IEEE.*

[16] “Nvidia,” www.nvidia.fr.