Received February 21, 2019, accepted March 4, 2019, date of publication March 11, 2019, date of current version April 1, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2904240

A Survey of Parallel Implementations for Model Predictive Control

KARAM M. ABUGHALIEH AND SHADI G. ALAWNEH, (Senior Member, IEEE)
Electrical and Computer Engineering Department, Oakland University, Rochester, MI 48309, USA

Corresponding author: Shadi G. Alawneh (shadialawneh@oakland.edu)

ABSTRACT Model Predictive Control (MPC) has its reputation since it can handle multiple inputs and outputs with consideration to constraints. However, this comes at the cost of high computational complexity, which limits MPC to slow dynamic systems. This paper provides an overview of the available methods to accelerate the MPC process. Various parallel computing approaches using different technologies were proposed to speed up the execution of MPC, some of these approaches are focused on building dedicated hardware for MPC using field programmable arrays (FPGA), and others are focused on parallelizing MPC computation using multi-core processors (CPUs) and many-core processors (GPUs). The focus of this survey is to review the available methods for accelerating MPC process. A brief introduction to the theory of MPC is provided first followed by a description of each approach. A comparison between the different methods is presented in terms of complexity and performance followed by a valid application for each approach. Finally, this paper discusses the challenges and requirements of MPC for future applications.

INDEX TERMS Control systems, FPGA, GPU, multi-core processing, quadratic programming.

I. INTRODUCTION

Model Predictive Control (MPC) is a control method based on optimization of a cost function subject to plant dynamics and constraints [1]. MPC was developed to control power plants and petroleum refineries [2], now MPC can be found in a wide range of applications involving mechanical and electrical systems in addition to industrial process control, such as vehicle traction control [3], suspension [4], automotive powertrains [5], autonomous driving [6], agricultural greenhouse [7], aerospace [8], and revenue management [9].

MPC strategy depends on solving an online optimization problem at every sampling time to minimize a specified cost function over specific horizon. The cost function defines the control steps overhead to reach a certain goal. The result is a sequence of optimal control steps for the whole horizon, only the first control step is applied to the system and the whole process is repeated for the next sampling time. The optimization problem of the cost function forms a quadratic programming (QP) problem since the cost function is in quadratic form in most cases or a linear programing problem (LP).

LP could be solved using simplex method. The simplex method is an optimization algorithm for solving Linear Programming (LP) problems presented in 1947 [10]. The method searches the feasible point one by till it match the criteria of optimization, the iteration is done on the vertices of the simplex area. A more efficient version known as the revised simplex method was developed later. The simplex method could be found in many commercial LP solvers. Solving the QP problem for every sample takes long time and it involves complex computations. This has limited the range of applications to which MPC can be applied; such applications should have slow dynamics and slow sampling time. One way to enhance the speed of MPC is to solve QP problem offline starting at an initial state and then providing the solution as a lookup table [11]. This method is known as ‘‘explicit MPC’’ but it is limited to small problems with no more than five states.

Due to the scientific and technological advancement and the industrial development, the requirement on control is getting higher and higher. Optimization provides a powerful control technique in order to achieve better performance within certain constraints. Optimization in MPC or other control strategies is restricted by increasing number of factors and constraints. The constraints could be physical such as tanks sizes and actuators characteristics or related to safety, energy consumption, etc. Combining high requirement and more complicated constraints is a big challenge for optimization based control strategy such as NMPC.

The associate editor coordinating the review of this manuscript and approving it for publication was György Eigner.

MPC technology achieved great success and gained a good reputation as an advanced control strategy that can deal with online optimization control for constrained MIMO systems in systematical pattern, but the bottle-necks of the current algorithms limit MPC to slow dynamics application, most of the existing model predictive control algorithms are designed to work for slow dynamic processes. In general, MPC algorithms functions better on high performance computers which makes it practically difficult to generalize its application into a wide scope of fields and scenarios. Ideally, a typical industrial application of MPC results in a heavy computation burden and considerable calculation time spans that define a significant computation complexity. In this regard, MPC emerges and only suitable for slow dynamic processes that have a larger sampling period. Consequently, the approach is also not conducive for an execution environment configured with low performance computing technologies. For this reason, the application of the MPC is commonly applied in the process industry. Another challenge for MPC relates to the fact that algorithms mainly focus on linear processes. As such, related technologies and software application are designed to handle linear systems. Despite the existence of non-linear systems developed by software vendors, the bulk of model predictive systems consist of linear technologies and capabilities. The existing non-linear systems are also limited to industrial processes that demonstrate weak nonlinearity and thus the application of the model predictive control in strongly non-linear processes or industries remains rare. The reason behind this is that it is extremely difficult, expensive, and time consuming to develop and implement the non-linear model predictive control.

To the best of our knowledge, a survey of parallel implementations for MPC does not exist in the literature. Thus, the main objective of this paper is to fill in this gap.

In this paper, we review various parallel computing approaches that can speed up the computations of MPC. These approaches will be categorized in three different groups: multi-core processors accelerated MPC, GPU accelerated MPC, and FPGA accelerated MPC.

The remainder of this paper is organized as follows. Section II provides a general overview of model predictive control, and then the three categories of MPC acceleration approaches with a summary table of their performance speedup enhancement are reviewed in Section III. In Section IV, a discussion of the challenges and requirements of MPC for future applications is presented. This section will also discuss the advantages and disadvantages of using each parallel computing approach to accelerate the computations of MPC. Finally, the paper is concluded in Section V. In order to help the reader navigate through the paper, Fig. 1 provides an overview of the overall organization of the paper in form of a chart.

II. MODEL PREDICTIVE CONTROL

Model Predictive Control (MPC) computes the optimal control input for a system, by using a model of the system,

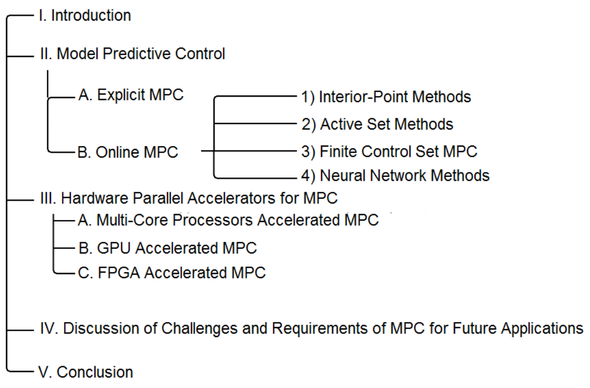
I. Introduction
II. Model Predictive Control
A. Explicit MPC
B. Online MPC
1) Interior-Point Methods
2) Active Set Methods
3) Finite Control Set MPC
4) Neural Network Methods
III. Hardware Parallel Accelerators for MPC
A. Multi-Core Processors Accelerated MPC
B. GPU Accelerated MPC
C. FPGA Accelerated MPC
IV. Discussion of Challenges and Requirements of MPC for Future Applications
V. Conclusion

FIGURE 1. The organization of the paper.

which describes system dynamics, to predict the future states and output. MPC succeeded as a control algorithm due to the following factors. 1) The use of plant model in the algorithm enables MPC to consider all the plant dynamics. 2) MPC works over a receding horizon, which means that the future effect of control can be anticipated with consideration to plant behavior, leading to better control toward the desired trajectory. 3) MPC calculations consider input and output constraints, leading to optimal constrained controlled for the process. This is the most desired feature in MPC.

In MPC formulation three main approaches were used to build the model of the plant under control, finite Impulse response and step response models [12], transfer function models [13] and state-space models [14], [15] which have seen a growing popularity in the last two decades.

The formulation of MPC problem for a plan model presented by the state space equations for a plant that has m inputs, q outputs and $n1$ states:

$$x_m(k+1) = A_m x_m(k) + B_m u(k) + B_d \omega(k) \tag{1}$$

$$y(k) = C_m x_m(k) \tag{2}$$

where x_m is the state variable, u is the manipulated variable (input variable), $\omega(k)$ is the input disturbance related to a zero-mean white noise sequence $\epsilon(k)$ as in equation 3, y is the system output, and A_m is the state matrix with dimension $n1 \times n1$, B_m is the input matrix with dimension of $n1 \times m$, B_d is the disturbance matrix with dimension of $n1 \times m$ and Cm is the output matrix of dimensions of $q \times n1$.

$$\omega(k) - \omega(k-1) = \epsilon(k) \tag{3}$$

starts by defining the difference equation for the state variable x_m as follows:

$$x_m(k) = A_m x_m(k-1) + B_m u(k-1) + B_d \omega(k-1) \tag{4}$$

Then by defining $\Delta x_m(k) = x_m(k) - x_m(k-1)$ and $\Delta u(k+1) = y(k+1) - y(k)$, subtracting 3 from 1 leads to:

$$\Delta x_m(k+1) = A_m \Delta x_m(k) + B_m \Delta u(k) + B_d \epsilon(k) \tag{5}$$

Now $y(k)$ can be related to the defined variable $\Delta x_m(k)$ as follows:

$$\begin{aligned}\Delta y(k+1) = C_m \Delta x_m(k+1) &= C_m A_m \Delta x_m(k) \\ &+ C_m B_m \Delta u(k) + C_m B_d \epsilon(k)\end{aligned} \tag{6}$$

where $\Delta y(k+1) = y(k+1) - y(k)$.

Introducing the new variable $x(k) = [\Delta x_m(k)^T y(k)^T]^T$, the state space model in matrix form for MPC is:

$$\begin{aligned}\begin{bmatrix}\Delta x_m(k+1) \\ \Delta y(k+1)\end{bmatrix} &= \begin{bmatrix} A_m & o_m^T \\ C_m A_m & I_{qxq}\end{bmatrix}\begin{bmatrix}\Delta x_m(k) \\ y(k)\end{bmatrix} \\ &+ \begin{bmatrix} Bm \\ C_m B_m\end{bmatrix}\Delta u(k) + \begin{bmatrix} B_d \\ C_m B_d\end{bmatrix}\epsilon(k)\end{aligned} \tag{7}$$

$$y(k) = \begin{bmatrix} o_m & I_{qxq}\end{bmatrix}\begin{bmatrix}\Delta x_m(k) \\ y(k)\end{bmatrix} \tag{8}$$

where $I_{q\times q}$ is the identity matrix with dimensions $q \times q$, and o_m is $q \times n_1$ zero matrix. For simpler notation equations 7 and 8 are rewritten in the following form:

$$x(k+1) = Am(k) + Bu(k) + B_d \omega(k) \tag{9}$$

$$y(k) = Cx(k) \tag{10}$$

where A, B and C are the system matrices.

Having the model in 9 and 10 and the measured state variable at time k_i the future predictions of the system with the future control inputs could be calculated. The future control steps and state variables are denoted by:

$$\Delta u(k_i) \;\; \Delta u(k_i+1) \cdots \;\; \Delta u(k_i + N_c - 1) \tag{11}$$

$$\begin{aligned} &x\ (k_i+1|\,k_i),\ x\ (k_i+2|\,k_i), \\ &\cdots, x\ (k_i+m|\,k_i), \cdots,\ x\ \left(k_i+N_p|\,k_i\right)\end{aligned} \tag{12}$$

where N_c is the control horizon which defines the number of control steps applied to the system, and N_p is the prediction horizon which defines the number of future state predictions to be computed. The notation $x(k_i+m|k_i)$ points to the state variable predicted at time k_i+m for the given measured model state at k_i.

Based on the state space model, the future state variables are calculated sequentially, defining the vectors ΔU and Y as follows,

$$\Delta U = [\Delta u(k_i)^T \;\; \Delta u(k_i+1)^T \cdots \;\; \Delta u(k_i+N_c-1)^T]^T \tag{13}$$

$$\begin{aligned} y = &[y(k_i+1|k_i)^T y(k_i+2|k_i)^T y(k_i+3|k_i)^T \\ &\ldots y(k_i+N_p|k_i)^T\,]^T\end{aligned} \tag{14}$$

the expectation operator is expressed as:

$$Y = Fx(k_i) + \phi \Delta U \tag{15}$$

where

$$F = \begin{bmatrix} CA \\ CA^2 \\ CA^3 \\ \vdots \\ CA^{N_p}\end{bmatrix}; \tag{16}$$

$$\phi = \begin{bmatrix} CB & 0 & 0 & \ldots & 0 \\ CAB & CB & 0 & \ldots & 0 \\ AS^2B & CAB & CB & \ldots & 0 \\ \vdots & & & & \\ CA^{N_p-1}B & CA^{N_p-2}B & CA^{N_p-3}B & \ldots & CA^{N_p-N_c}B\end{bmatrix} \tag{17}$$

For a given set of target trajectories $R_s^T = \begin{bmatrix} r_1 & r_2 & \ldots & r_{N_p}\end{bmatrix}$ the goal of MPC is to minimize the error between the system output and the given trajectories vector $(Y - R_s)$ using the optimized control steps (ΔU) within a certain constraints on the output and control steps. Now the MPC problem is expressed in the following form:

$$\begin{aligned} &min\ J(\Delta U) \\ &\text{s. t. } Y = Fx(k_i) + \phi \Delta U \\ &\quad U_{min} \le U \le U_{max} \\ &\quad \Delta U_{min} \le \Delta U \le \Delta U_{max} \\ &\quad x_{min} \le x(k_i) \le x_{max} \\ &\quad Y_{min} \le Y_{t+k} \le Y_{max}\end{aligned} \tag{18}$$

where J is the cost function over the receding horizon of P samples, $Y$is the constrained output vector of the systems described by the constrained state x and controlled by constrained input ΔU control steps vector, The cost function J can be formed in different ways to penalize certain terms or control actions, the usually used form is:

$$\begin{aligned} J = &(R_s - Fx(k_i))^T (R_s - Fx(k_i)) \\ &- 2\Delta U^T \phi^T (R_s - Fx(k_i)) + \Delta U^T (\phi^T \phi + \bar{R})\,\Delta U\end{aligned} \tag{19}$$

where $\bar{R}$ is a $N_c \times N_c$ diagonal matrix used for tuning, The control and prediction horizons are illustrated in Fig. 2. In order for the MPC process to achieve its goals, MPC controller, which is illustrated in Fig. 3, should follow a specific strategy, which is described below:

1) At each instant k_i, the current state of the system $x(k_i)$ is sampled and used as the base to compute the future outputs of the system for a horizon of N_p.
2) The set of optimal control signals ΔU is obtained by optimizing the cost function J to drive the predicted output to the desired reference R_s. The cost function includes the error from reference deviation penalty in addition to other costs like control effort. Here, the problem of MPC becomes a quadratic programing (QP) problem. An important assumption is the resulting QP here is feasible which means that there is a solution to the QP under the desired constraints.

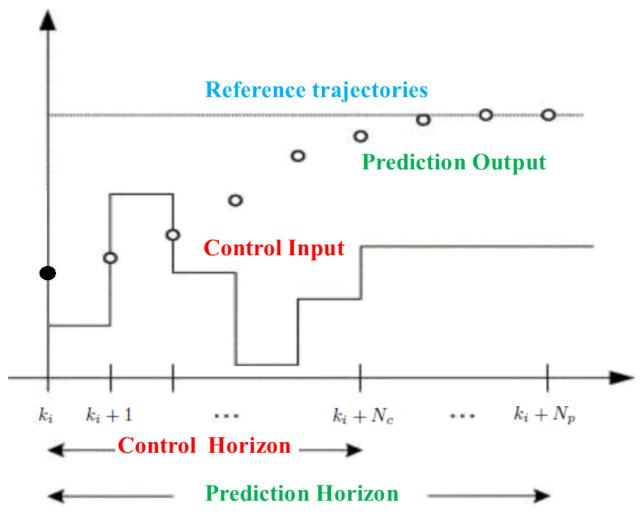

FIGURE 2. MPC receding horizon strategy [16].

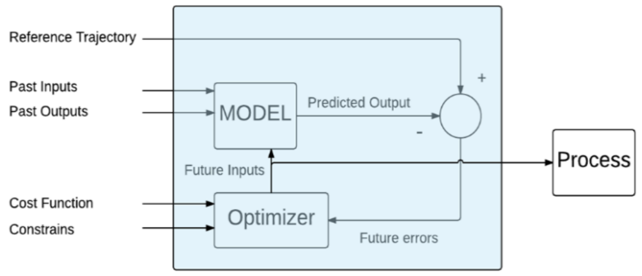

FIGURE 3. MPC structure.

3) The first control signal $u(k_i)$ is taken as the MPC output to the process, while the other control signals are dismissed. Then the whole process is repeated again at the next sampling step.

Various strategies are used to enable MPC to be used for fast sampled systems; the following subsections provide an overview of the main methods that are presented in the literature.

A. EXPLICIT MPC

The goal of explicit MPC is to get off the solution of an optimization problem online, and carry the computational burden offline. The optimal control actions are determined as a function of the state for a given predefined set, known as explicit solution, which can be computed by means of parametric programming methods. Explicit MPC offers extremely high sampling rates, making it possible for MPC to be used for high speed systems. Explicit MPC offers advantages like closed-loop feasibility, stability, and robustness. Another important advantage for explicit MPC is its implemented using a look-up table which can be easily implemented in hardware or software. Explicit MPC has also limitations, the computation time and solutions size grow in the worst case exponentially with the problem size. This limits explicit MPC to relatively small problem dimensions.

The limitation of explicit MPC increased the interest to come up with new methods to approximate explicit solutions. Approximate explicit MPC strategies stand in the middle between the advantages of explicit MPC and the complexity of the explicit solution. These methods try to keep the advantages of explicit MPC while eliminating the main disadvantages. As in [17] where the authors reduced the complexity of MPC by deriving a minimal representation of the explicit solution. In [18], they used region elimination to reduce the number of regions based on stability existence using neighboring control law.

B. ONLINE MPC

Online MPC methods can handle any problem size. In the cost of higher computation and memory requirement. It is the general MPC case, where the problem is solved online without any prior online calculations. Different methods were presented to accelerate online MPC computations. Many methods adopted the idea of taking advantage of sparsity nature of the optimization problem presented in MPC in order to reduce the computation complexity. Other methods focused on reducing the number of optimization iterations by finding an initial point to start the optimization. In the following subsections, some techniques are discussed given the optimization method used in each one.

1) INTERIOR-POINT METHODS

In Interior-Point Methods (IPMs), the effort for finding the optimal solution for a problem grows polynomially with the size of the optimization problem and the termination condition [19]. IPM iterations are computationally expensive; this is because of the Newton step computations, where the matrices have to be factorized at each iteration. Computation complexity reduction could be achieved by utilizing the structure and sparsity. The linear system in IPM could be solved using minimal residual (MINRES). MINRES is an iterative algorithm for solving linear systems with indefinite symmetric matrices suitable to run in parallel.

2) ACTIVE SET METHODS

Active Set Methods (ASMs) require a larger number of iterations, but compared to IPMs these iterations are less expensive. The active set method can be used either for the sparse or dense structures. Active set methods can utilize a feasible starting point. The main computational effort in an active set process is the solution of the Karush-Kuhn-Tucker (KKT) system for computing the search step direction. Therefore, most of the research effort is concerned with the efficient computation of the step direction [20]. Other approaches for fast MPC are proposed in literature, like the dual gradient projection method for QP [21] and the fast gradient method [22].

3) FINITE CONTROL SET MPC (FCS-MPC)

FCS-MPC approach is used for application with finite number of control actions, as in power converters. Such system has a limited number of switching states. Therefore, MPC optimization problem has a fixed number of predictions for the system. Then, to solve the problem, the perditions cost functions are evaluated and the one with the minimum cost function is used. FCS-MPC is widely used in power converter and drive applications [23]–[25].

4) NEURAL NETWORK METHODS

The neural network methods are now thought to be a powerful approach for online optimization. Neural network based MPC might use plant data as a training set to predict the plant output. The training set could be a real measurement of inputs and outputs for the plant using an open loop control or conventional controllers, as in [26] without the need to know anything about the dynamics of the plant. Other approaches use the primal-dual neural network [27] to solve the QP in MPC as in [28]–[31]. The following section presents an extra level of MPC acceleration using hardware accelerators.

III. HARDWARE PARALLEL ACCELERATORS FOR MPC

The traditional way software works is based on serial computations. That means the instructions that solves the problem are executed sequentially one after the other, once at a time on a single processor. The first steps toward parallelism were in the instruction level (ILP) such as pipelining, superscalar, very long instruction Word (VLIW), single Instruction Multiple data (SIMD). In addition to other areas focusing on longer pipelining, larger registers and cache. These techniques rely on the possibility to increase circuit density as Moore's Law suggested. Increasing the number of transistors in the circuit led to a higher frequency and better performance but processor clock speeds began to flatten due to the limitations imposed because of power and heat that limits clock frequencies.

High-Performance Computing (HPC) refers to a class of computing where a complex problem or a problem with huge amount of data to process is broken down to a smaller pieces; tasks wise or data wise; using the software level then each piece is processed by a multi-core CPU units. Having multiple cores lets the CPU process multiple chunks of data or multiple tasks at one time. Getting more cores solves the problem of struggling to get higher speed frequencies, even with slower multi-core processors the overall throughput is higher than high speed frequency single core where problem parallelization is possible. Increasing the number of processor cores is a cost effective way to increase the performance even that single core processors are cheaper.

Fig. 4 shows the differences between the CPU and GPU architectures. The many-core processors like GPUs, takes the high-performance parallel computing one step further. Focusing more on throughput of large number of working threads executed by large number of cores. These cores are much simpler than the ones in general purpose CPUs. In 2016, the ratio of peak floating-point calculations throughput between GPUs and multi-core CPUs reach 10.

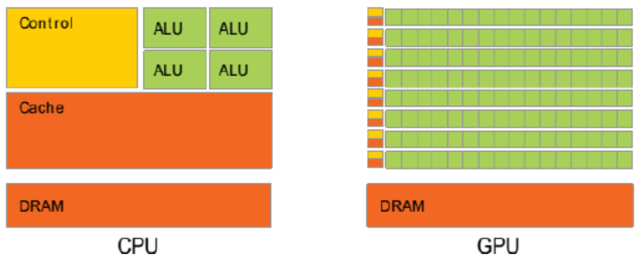

FIGURE 4. CPU vs GPU Architecture [33].

The software for GPU's should be written using a large number of parallel threads [32]. So the hardware could effectively utilize the work to be done and compensate for the latency due to memory access, in GPU's a cache memory is used to minimize the number of times that the threads need to access the DRAM. This approach is referred as throughput-oriented design, where a large amount of data is expected to be processed by a large number of threads while each individual thread is allowed to take long time for execution. On the other hand, CPUs are strongly designed to minimize the latency for a single thread by reducing the memory access time using larger sizes for cache memories. The design of the arithmetic logic units and data registers is also harnessed to reduce thread latency. The trade-off here; due to this implementation of many optimization techniques, is a larger chip area and power. This design approach is referred as latency-oriented approach.

Another form of parallel processing is the use of Field-Programmable Gate Array (FPGA). As a high level definition, FPGAs are reconfigurable chip using pre-built logic blocks. FPGAs are parallel by nature, different processing operations are built using dedicated resources on the chip while designing, so they do not have to compete for the same resources while operating. As a result the performance of any part of an application built on FPGA is not affected when extra processes are added. FPGA design is cost effective but at the same time a developer might be limited by the number of available logic units in the chip.

The recent MPC research status shows a huge interest in MPC to solve more complex problems in increasing number of fields. However, new approaches and new methods are required to overcome the limitations of the existing MPC theory. Studies on efficient MPC structures and strategies have been reported in order to handle large scale optimization problems as in traffic management and wastewater treatment [34], [35] such problems have complex models and huge number of variables. One way to reduce the computational complexity in large-scale system, is using distributed structure [36], this strategy decomposes the large-scale MPC problem into smaller scale problems. Another group of studies focused on making improvements for the standard optimization algorithm as in [37]. More efficient performance for real-time applications is required and therefore the research became more interested in hardware accelerated

MPC such as programmable logic controller (PLC) [38], FPGA, multi-core CPU and GPUs as listed in this survey.

MPC complexity is a function of the number of states, constraints, the length of the control, and prediction horizons. Running MPC on machines with high resources for a slow dynamics in the order of hundreds of milliseconds to second like in industry process is acceptable, but for processes with fast dynamics in the order of milliseconds the optimization result should be ready in a short sampling time of the order of millisecond.

There has been a lot of activity over the last two decades on parallel implementations for MPC. A quick reference for the reviewed work in terms of the used hardware accelerator, the main focus of the work and the performance enhancement is shown in Table 1.

The following subsections discuss the main hardware accelerators that are used to improve the computations of MPC. These subsections also describe the related work review for each one.

A. MULTI-CORE PROCESSORS ACCELERATED MPC

The use of two or more Central Processing Units (CPUs) in a system to share the workload is a well-known approach to solve problems in parallel in a reduced time. This section is devoted to the review of the literature on MPC acceleration using multi-core processors. Dividing a MPC problem into sub-problems having a smaller scale [39] or lower complexity [40]–[42] is the main idea behind MPC parallelism. Smaller and simpler problems can be solved with the given resources in parallel in a faster and more efficient way. The solution for the MPC is the combination of all the generated sub-problems.

Newton step computations optimization gained a lot of interest since it is the main computational effort in interior point or active set methods. Soudbakhsh and Annaswamy [40] introduced an extended Parallel Cyclic Reduction algorithm to reduce the computation to a simpler system of equations that can be solved in parallel. The computational complexity of this algorithm is reported to be O (log N), where N is the prediction horizon. In [39] a tailored, non-iterative parallel algorithm for computing the Newton step using the Riccati recursion is proposed. The choice of Riccati recursion is because it can be solved in parallel. The authors reduced the MPC problem into smaller problems with a shorter prediction horizons, more details on problem formulation to solve (KKT) system and dividing could be found in [39]. The proposed algorithm solves the Newton step arising in MPC problems in O (log(N)). Decomposing MPC to smaller sub-problems is also presented by Kögel and Findeisen [42] using an alternative directions multiplier. MPC problem is decomposed along the time axis into smaller sub-problems. OpenMPI is used to implement the parallel computation. The authors tested the implementation on Intel Core 2 Quad Q6600 CPU with 2.4GHz and an Intel Xeon X5675 CPU with six cores at 3.46GHz. The reported speedup on X5676 is 4.3x when using the six cores compared to single core on the same processor and 3.1x speedup on Q6600 four cores compared to the single core on the same processor. Splitting method is also used in Ferranti and Keviczky [41]. The proposed approach combines Alternating Direction Method of Multipliers (ADMMs) and Dual Fast Gradient (DFG) methods. The authors compared their work to the one in [43] and reported a speedup of 230x.

In a different approach Buchner and Skworcow [44] used the dSpace multi-processor system for a three-tank control problem using MPC with the Danzig QP-Solver. The authors divide the tasks of MPC into six smaller tasks, and analyzed different scheduling setups the best achieved speed up was 1.8x. The communication overhead between the real-time processors made the implemented algorithm slower than the same one implemented on a single core. The authors recommend reducing the communication between the processors or using parallel QP algorithm [45].

B. GPU ACCELERATED MPC

Graphical Processing Units (GPUs) have massively parallel architecture. Such architecture provides high floating point operations performance for certain numerical operations. This comes at the cost of more algorithms rewriting to suit the nature of GPU, and more restriction on the workload that can be accelerated by the GPU because not all numerical operations are equally enhanced using GPUs.

Two programing models are available to run a heterogeneous system; where CPUs and GPUs cooperate to process certain computations, Compute Unified Device Architecture (CUDA) [33] and Open Computing Language (OpenCL) [46]. CUDA is developed by NVidia and works only on NVidia GPUs while OpenCL can work on GPUs from different vendors. OpenCL was initiated by Apple Inc. and then maintained by the Khronos Group as an industry standard API. This section is devoted to the review of the literature on MPC acceleration using GPUs.

The main enhancement GPU processing provides is related to matrix-matrix, matrix-vector and vectors-vector operations. This enhancement can be utilized using a well optimized linear algebra libraries like cuBLAS as in [47] or by writing special kernels as in [48]–[50]. Gade-Nielsen *et al.* [48] implemented a GPU accelerated linear programing solver for interior point method to solve linear optimization problems with inequality constraints. In their work, the authors implemented different solvers using Matlab and Matlab GPU in addition to CUDA C version.

Their CUDA C implementation on the NVIDIA Tesla C2050 GPU showed a speedup of 6x compared to Matlab version running on Intel Core i7 930 CPU. While the built-in GPU accelerated Matlab function showed a speed up of 2x.

Yu *et al.* [49] implemented and analyzed the performance of Parallel Quadratic Programing (PQP) and Alternating Direction Method of Multipliers (ADMM) solvers to solve MPC QP, both methods are gradient based. In their work, they used the NVIDIA Jetson TX1, for single thread implementation and multithreaded implementation where the

TABLE 1. Performance enhancement and speedup of the reviewed work.

Work	Hardware Accelerator	Focus	Performance Enhancement
[40]	Multi-Core CPU	extended Parallel Cyclic Reduction algorithm	computational complexity O (log N)
[39]	Multi-Core CPU	computing the Newton step using the Riccati recursion	Newton step computational complexity O (log N)
[42]	Multi-Core CPU	alternative directions multiplier	4.3x using six cores compared to single core on X5676 and 3.1x speedup four cores compared to the single core on Q6600 processor.
[41]	Multi-Core CPU	inexact numerical optimization algorithms and operator splitting methods	230x compared to [45]
[48]	GPU	interior point method	6x compared to Matlab version
[49]	GPU	Alternating Direction Method of Multipliers (ADMM)	46.6x compared to a single thread
[51]	GPU	hybrid data-parallel and problem-parallel approach for nonlinear system.	GPU execution time less than 250ms while CPU start increasing exponentially for large data
[47]	GPU	accelerated dual proximal gradient algorithm	22.6x compared to Intel i5 machine with 8GB 250 of RAM
[50]	GPU	parallel Gaussian elimination	30x compared to AMD Athlon64 3000+, DDR400 1GB
[52]	GPU	parallel version of Nelder-Mead simplex algorithm	2.7x compared to Intel 2.98 GHz Core(TM) 2 E7500 CPU
[53]	FPGA	interior point using customized floating point format	required time for control action is less than 200us
[54]	FPGA	FOR loops optimization	2x compared to Matlab runtime
[55]	FPGA	optimize the gradient and hessian on custom hardware	reached a sampling time of 1ms
[56]	FPGA	Parallel Move Blocking (PMB) algorithm computing architecture	faster sampling compared to the standard MPC
[57]	FPGA	multiplexed MPC (MMPC)	2.25x compared to Intel Core2 Q8300 with 3GB of RAM
[58]	FPGA	Gradient Method (FGM) and alternating direction method of multipliers (ADMM) in a fixed-point arithmetic format.	sampling rates in the region of 700kHz with Spartan 6 and 40kHz to 200kHz for different Virtex 6 devices
[59]	FPGA	a dual gradient projection (DGP) algorithm	4x compared to floating point implementation
[60]	FPGA	fast gradient method	2x to single core implementation
[61]	FPGA	used optimality conditions based on Hamiltonian	3-5x sequential computation on FPGA 1.4-2.6x faster compared to Intel Core i5
[62], [63]	FPGA	parallel Euler integration	21x compared to a sequential implementation on a Microblaze processor
[64]	FPGA	a general-purpose microprocessor in addition to a matrix coprocessor	30x compared to the Motorola's 32-b MPC 555 processor running at 40MHz

baseline comparison was the ARM Cortex A57 in TX1. Based on the analysis, the authors decided that the most computationally intensive routine inside the iterations is matrix-vector multiplication and therefore they developed more efficient SGEMV kernel that outperformed cuBlas library on the TX1 by a factor of 2.3x. Overall, the authors were able to achieve 46.6x speed up over the single thread CPU for ADMM and 2.7x speed up over the optimized OpenBLAS version, while for PQP the peed was 41.2x over the single threaded CPU version and 4.2x over the OpenBLAS version.

Gang and Minggunang [50] proposed MPC acceleration on GPU by solving the equations using parallel Gaussian elimination the simulation is conducted on AMD Athlon64 3000+, DDR400 1 GB and NVIDIA GeForce 8800GT. The reported acceleration using the system in 20 with different horizon lengths was valid only for prediction horizon greater than 42 the speed up was 1.4x and for N = 682 the speed up was 30x.

$$\begin{bmatrix}\dot{x_1}\\ \dot{x_2}\\ \dot{x_3}\end{bmatrix}=\begin{bmatrix}0 & 1 & 0\\ 0 & -1.0860 & 4.287925\\ 0 & -1.0909 & -1.4545\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix}+\begin{bmatrix}0\\ 0\\ 3.6363\end{bmatrix}u \tag{20}$$

Sampathirao *et al.* [47] proposed GPU accelerated MPC for the application of controlling drinking water networks. The authors propose an accelerated dual proximal gradient algorithm for the solution of the optimal control problem resulted in the formulated stochastic MPC and compared the acceleration to CPU-based optimizer. The proposed control algorithm is applied to the data of drinking water network of the city of Barcelona. The authors used cuBLAS library to perform the matrix vector computations and compared it to interior-point solver of Gurobi. The used hardware was Intel i5 machine with 8GB of RAM for the CPU computations and NVIDIA Tesla C2075 for the GPU computations. The reported speed up was up to 22.6x.

Nonlinear MPC acceleration is studied in [51], the authors implemented MPC on the NVIDIA Jetson TK1 for autonomous navigation of a ground mobile robot. The authors used hybrid data-parallel and problem-parallel approach for nonlinear system. In their implementation the control vector is generated offline then copied into GPU, after that MPC optimization will look through this vector to find the control candidate sequence that entails the minimum cost. The focus of this implementation is adopting CUDA optimization techniques to get the best performance. The used robot is four-wheel Robotnik Summit XL with maximum linear speed of 1 m/s and radial speed of 0.5rad/s, which made the execution time of 250 ms for MPC, meets the real-time requirement. For speed up measurements, the authors ran the equivalent C++ code on Intel Xeon W3520 @ 2.67 GHz. The measurement variable was the data vector length, for smaller data the CPU showed faster execution because of the higher frequency of the CPU and higher available memory resources. However, the GPU performance is getting better for larger data size compared to CPU, where it is kept under 250ms and the CPU start increasing exponentially for data vector length of sizes more than one million.

The core of MPC is the optimization problem, in the literature a lot of work regarding accelerating optimization problem solvers has been reported without involving the MPC problem itself. Accelerating interior point method for linear programing is presented in [65] and [66]. In [65] a linear programing solver based on interior point method is proposed, the authors used GPU to accelerate tasks related to matrix assembly, Cholesky factorization, and forward and backward substitution. The used GPU is NVIDIA GeForce 7800 GTX compared to Intel Xeon 3.0GHz (1MB L2 cache, 8GB DDR2 dual channel memory, 400MHz effective memory clock cycle and 800 MHz FSB), the reported speed up was better than CPU version for large problems only. For a medium-sized Netlib LP the speed up was up to 1.4x, detailed speed up timing results are available in [65]. In [66], Smith *et al.* proposed matrix-free interior point method approach. The purpose is to avoid the explicit use of the problem matrices in the iterations; the iterations require only the results of Ax and $A^T y$ but not the matrix A where A is the matrix of constraint coefficients. The author compared their results to the simplex method and IPM. The used hardware is AMD Opteron 2378 (Shanghai) quad-core processors, 16 GB of RAM and a Tesla C2070 GPU with 6 GB of RAM. Their computational results revealed a maximum speed up of 10x on large sparse LPs compared to single core CPU and 6x using multi-core CPU implementation. Cholesky decomposition is the usual method of solving for the search direction. Acceleration using GPU for solving dense linear system has been also investigated in [67] and [68].

Simplex method for optimization parallelization using GPU was presented in [69]–[73]. In [69], CUBLAS library [74] is used to implement the Simplex method on Tesla S1070 GPU card to speed up the pivoting step in a multi-GPU structure, the reported speed up was up to 4x. In [70], a parallel implementation using CUDA of Simplex algorithm for dense LP problems was developed, the tested hardware setup is 3 GHz Xeon Quadro INTEL processor and a GTX 260 GPU with a reported speed up of 12.5x in double precision. Ploskas and Samaras [71] proposed parallel implementation of primal-dual exterior point Simplex Algorithm on GPUs using Matlab's parallel computing toolbox. It was the first parallel implementation for exterior point algorithm. The implementation showed significant computational result compared to sequential implementation of the same method and Matlab's sequential interior point method, the results obtained tests performed on randomly generated optimal sparse and dense Linear Programing (LP) problems in addition to a set of benchmarks. The reported speed up was 181x on dense LPs and 20x on sparse LPs.

Spampinato and Elstery [72] presented an implementation of the Revised Simplex Algorithm with NVIDIA CUBLAS and NVIDIA LAPACK libraries [75]. Their implementation speed up was 2.5x on large random LPs with 2000 variables compared to CPU implementation. Another parallel implementation of the Revised Simplex Algorithm on GPU is presented in [73]; to evaluate their implementations the authors measured the execution time against the corresponding serial implementation of the GNU Linear Programming Kit (GLPK). The reported speed up was up to 18x, the used hardware is Intel Core Duo E8400 and NVidia GeForce 9600 GT GPU.

C. FPGA ACCELERATED MPC

Modern FPGA technology provides inexpensive devices with high resources and keeps advancing. This opens the fields for applications with high resources demand. In addition, FPGA technology supports optimized use of parallel computations. This made FPGA capable of providing a performance level higher than other fixed architecture like microcontrollers. Because of FPGA technology advances, FPGA becomes suitable for MPC controller implementation, due to the high performance at MHz and low cost, even more for cost efficient industrial scale, MPC on chip system could be built on Application Specific Integrated Circuits (ASIC). This section is devoted to the review of the literature on FPGA based MPC implementations.

Interior point methods is implemented on FGPA with different optimization techniques like customized floating point [53], pipelining [53], [57], and for loops optimization [54]. Wills *et al.* [53] reported FPGA based interior point using customized floating point format (24 bits). Pipelining and parallel computation were used in matrix-vector multiplication. The hardware consists of Altera Stratix III EPSL150F115C2 FPGA, which is interfaced to external A/D and D/A circuit in order to read system status and output control actions. Working with fewer bits to represent a number decreases the arithmetic circuit size, power consumption and computation time. The author's experimental setup was the control of lightly damped resonant structure, with objective pf disturbance rejection. The reported required time for the control action is less than 200us. In [54] where the author focused on the FOR loops optimization including combining and parallelizing. Matlab runtime for two implementations of interior point method and active set method was recorded and compared to the FPGA implementation, the achieved speed up was up to 2x.

Jerez *et al.* [57] proposed MPC algorithm based on Multiplexed MPC (MMPC) [76]. The concept behind MMPC is that, it updates one input at a time, of a multi-input controlled plant. This results in a smaller QP at each sampling instant. The authors use MMPC as independent blocks to optimize each plant input independently and in parallel. FPGA implementation adopted (MINRES) method to solve Newton's step in IPM. This requires indefinite symmetric matrices which should be obtained in a prior step by row reordering. MINRES iterations are preferred because the required matrix-vector multiplication is easily parallelized on FPGA. The reported achieved frequency running the implementation on a Virtex 6 VSX 475T is up to 150 MHz. Benchmarked to Intel Core2 Q8300 with 3GB of RAM, 4MB L2 cache, and a clock frequency of 2.5GHz running Linux. The smallest achievable sampling time on the FPGA was 0.344 seconds while on the CPU it was 0.775 seconds.

Gradient based methods were investigated in [55] and [58]–[60]. Combining specialized hardware and optimizing techniques allowed Bleris *et al.* [55] to reach sampling time of 1 ms for a system with four states, control horizon of 3 steps and prediction horizon of 10 steps. The authors divided the tasks of the optimization problem into five main operations. The operations are initialization, gradient vector and hessian matrix computing, matrix inversion and Newton's iteration to find the search direction. Based on the tasks analysis, the authors decided to implement and optimize the gradient and hessian on custom hardware while using a matrix processor for the rest of operations.

In another work, Jerez *et al.* [58] provided custom architecture for Fast Gradient Method (FGM) and Alternating Direction Method of Multipliers (ADMM) in a fixed-point arithmetic format. The paper presents error analysis of both methods. In addition to balancing the minimum number of bits, solver iterations and the amount of required resources to get a satisfactory closed-loop performance. Implemented on a 400MHz clocked Xilinx's Virtex 6 chip and 230 MHz Xilinx's Spartan 6 chip the achieved sampling times are 1.8MHz for Virtex 6 and 700KHz for Spartan 6 using FGM while for ADMM it was 200kHz for Virtex 6 and 117kHz for Spartan 6.

Patrinos *et al.* [59] proposes a Dual Gradient Projection (DGP) algorithm tailored for implementation on fixed point hardware. The authors used minimum number of decimal and integer bits that guarantee convergence. The used hardware is 32-bit Atmel SAM3X8E ARM Cortex-M3 processing unit. Using QP problem size ranging from 10 to 60 variables and 20 to 120 constraints, the reported speed up was up to 4x compared to floating point implementation. In addition to reduced code size to the half. The authors implemented this approach as a control for spacecraft attitude tracking with reaction wheels actuators [77]. In order to reduce the computational load, a reduced control model is selected, in addition to a modified cost function with reduced prediction horizon.

Peyrl *et al.* [60] proposed parallel implementation using fast gradient method to solve liner QP in MPC. The authors provided fixed point FPGA and multi-core solutions and compared both implementations. The used hardware was Cyclon V FPGA and Freescale P4080 @1.2 GHz. Experiments showed that problems could be solved two times faster on the FPGA compared to single core implementation on the used multicore processor. The authors pointed that the overhead caused by the data transfer between cores requires big problems size to be ignored and make use of the parallelization.

Longo *et al.* [56] proposed Parallel Move Blocking (PMB) algorithm computing architecture that reduces the computational load of solving MPC optimization problem online using Move Blocking (MB) strategy. In MB, the predicted control trajectory is forced to remain constant over some steps; therefore, the degrees of freedom are reduced by fixing some optimization variables. In their implementation, the authors took the standard MPC problem and converted it into one with blocking allowing faster sampling compared to the standard MPC. The author's future work is to implement this approach on parallel hardware like FPGA and GPU.

Implementing Nonlinear Model Predictive (NMPC) control is investigated in [61]–[63], [78], and [79]. In [61] the authors used MATLAB HDL Coder and Vivado HLS for parallel implementation of MPC on FPGA. To solve the control problem in MPC the authors used optimality conditions based on Hamiltonian [80]. Fixed-point iteration method is used to make use of parallel processing of the FPGA to compute numerical integrations. The reported results show 3-5 times faster performance when using fixed-point iteration vs sequential computation on FPGA and 1.4-2.6 times faster compared to Intel Core i5 at 2.67GHz and 4GB memory. In Knagge *et al.* [78] proposed implementation of MPC on FPGA and Application Specific Integrated Circuit (ASICs). Nocedal and Wright used Sequential Quadratic Programing (SQP) method to solve MPC [81]. The authors made benefit of parallelism features in FPGA and ASIC devices in order to get efficient and high performance MPC controller. The authors did not report speedup results.

Vouzis *et al.* [64] presented a hardware architecture for a System on Chip (SoC) embedded real time implementation of MPC. The proposed system consists of a general-purpose microprocessor in addition to a matrix coprocessor devoted for matrix operation as addition, multiplication and inversion. The coprocessor performs and stores the matrix operation required by MPC process and communicates with the general microprocessor for reading the inputs and sending back the result. Arnold and Shuler [82] used VITO tool in the process of designing the matrix processor. The used language for the design is Verilog, and the used hardware is Xilinx FPGA ML401 board equipped with XC4VLX25-FF668-10C Virtex-IV FPGA. The reported speedup compared to the Motorola's 32-b MPC 555 processor running at 40 MHz is up to 30x.

Xu *et al.* [79] presented a fast NMPC algorithm implemented on FPGA that employs a Particle Swarm Optimization (PSO) with the penalty function approach is used as the embedded optimization algorithm. PSO is stochastic optimization technique inspired by the social behavior of bird flocking and fish schooling. The system is seeded with a population of random solutions then searches for optimal by updating new generations. The nature of PSO provides parallel capabilities, which is necessary for real time operation, and suitable for the parallel architecture of FPGA. The authors reported the timing to be 0.8s for the system for engine to track the reference speed in their experimental setup. PSO is also used in [83] in a control approach under actuated overhead crane system for limiting the swings. The error in the position of the crane is optimized using PSO. The authors reported successful control in terms of limiting the payload deflection and reducing residual vibration. In terms of timing the proposed approach recorded faster settling time than Fuzzy Gain Scheduling (FGS) and zero vibration derivative (ZVD) methods, for more details on the results refer to [83].

[62], [63] a parallel real-time implementation of NMPC with constrains on FPGA proposed as controller for 3-dimensional trajectory planning in a fixed wing Unmanned Aerial Vehicle (UAV). Two IPs were implemented one to interface with the sensor and the second id for NMPC, parallel Euler integration is implemented in the NMPC IP. The implementation on Xilinx Spartan 3E 1600 FPGA required 5.1ms compared to 111ms for the sequential implementation on a Microblaze processor.

IV. DISCUSSION OF CHALLENGES AND REQUIREMENTS OF MPC FOR FUTURE APPLICATIONS

Implementing MPC on parallel processors is not an easy task. MPC compatible implementation requires finding concurrency in the algorithm then dividing it into sub-processes that can be done in parallel. These tasks are profiled and analyzed in order to achieve the best optimization and know which should be accelerated. MPC structure involves a lot of linear algebra operations applied on large matrices, like matrix-matrix multiplication, matrix-vector multiplication, matrix inverse and matrix decomposition in addition to operations like reduction to find the minimum element or maximum element in a vector. CPU linear algebra libraries like BLAS and LAPACK provides a lot efficient and well optimized matrix operations, these libraries are extended to GPU with high performance and efficiency as in CUBLAS and MAGMA.

This structure of MPC is suitable for the both approaches of algorithm acceleration, the problem-parallel approach and the data parallel approach. in data parallel approach implies that the same operation is executed on multiple data sections as the case in matrix multiplication for example. In problem parallel approach the MPC problem is divided into smaller problems with the same size solved using the same algorithm steps. Combining both approaches on hardware accelerators provides more utilization of the available resources in these accelerators.

Decomposing MPC into smaller sub-problems on GPUs, by dividing the prediction horizon will suffer from the latency of working threads and the overhead of dynamic parallelism where the thread is in need to launch other threads to execute different operations required by MPC. On the other hand this implementation on FPGA will be limited by the number of programmable units and available memory. The reason is the need for multiple copies of the sub-problem solver unit which involves multipliers and other units.

FPGA implementations reported promising results but faces many challenges. Problem size in terms of the number of variables form the biggest challenge. The FPGA is a resource limited system especially if the division and root operations are extensively required as reported in [54].

TABLE 2. Hardware accelerators overview.

Accelerator	Key Advantages	Key Disadvantages
Multi-Core CPU	• Availability of well-optimized linear algebra libraries • Available for embedded implementations of MPC for a low cost	• Processors communication overhead • Suitable for small to mid-size problems
GPU	• Availability of well-optimized linear algebra libraries • Hardware is optimized for massively parallel computing, which makes it suitable for large MPC problems	• Overhead of memory transfer between the host processor and the GPU • Overhead of kernels creation, especially when converting MPC to smaller scale problems
FPGA	• Dedicated resources for multiple operations; different tasks of MPC can be performed in parallel with high frequency • Energy efficient compared to GPU and CPU	• Problem size in terms of the number of variables form the biggest challenge. The FPGA is a resource limited system especially if the division and root operations are extensively required • FPGA implementation for MPC is application specific leading to loss of flexibility, any model changes require new design and route operations

FPGA implementation was outperformed by Pentium CPU due to the usage of straightforward Gaussian elimination method to solve a linear system of matrices. On the other hand fitting the design on the FPGA is big challenge too, 93% utilization is achieved in [84] to achieve a maximum frequency of 88.2MHz using Altera DE2 development board. Other points to present here are FPGA implementation for MPC is application specific leading to loss of flexibility, any model changes require new design and route operations. In systems where FPGA is used with a co-processor of the communication between the FPGA and the co-processor is a negligible overhead for a small problem size but should be considered for large problems especially if the communication is performed for each iteration, this applies also for MPC parallel implementation on multi-core processors. One of the used techniques to improve performance in FPGA MPC implementation is using fixed or custom numbers format. When using interior point method, MPC solution depends on the precision and the criteria used in the convergence test.

Implementing MPC on a GPU has its own challenges also, implementing parallel kernels in a well-optimized way is a hard task. Kernel calling is considered an overhead especially for simple operations where the time of the processing is less than the kernel call delay. Developers avoid kernel overhead by combining multiple operations in a single kernel to maximize the GPU utilization. This technique is known as kernel fusing. GPUs are optimized for throughput, making them suitable for large problem size with a noticeable speed up that can't be achieved for a smaller problem. On the other hand, problem size presents another challenge; the data copying overhead between the host CPU and the GPU. Developers are encouraged to minimize the copying as much as possible in the process of solving MPC problem especially with the iterations involved. In some cases a developer might use more than one library to perform a specific operation and here the conversion between the different types of data will not be an easy task, however a high level library like Thrust [85] should avoid that inconvenience. Table 2 provides a summary for the key advantages and disadvantages of the reviewed hardware accelerators for MPC.

V. CONCLUSION

The acceptance of the model predictive control system in new emerging fields has demonstrated the various challenges and limitations inherent to the traditional approaches to model predictive control. In light of identified challenges, different actors in a variety of industries have come to the realization that the MPC is applicable in a variety of industries. Scholars and researchers should strive to bridge the gap between theory and practice. In terms of coming up with more suitable strategies for parallel computing to achieve real-time performance.

In this survey, different approaches were presented from the literature for efficient MPC acceleration in terms of new ideas and methods in the theory and structure of MPC. This survey presented the reported speedup for each study. However, application areas are in continuous growth with a higher demand on more efficient algorithms that meet real-time requirements. This has opened the doors for hardware accelerated implementations, where fast MPC ideas are implemented to get even faster performance.

Hardware accelerators could be very efficient in the search operation in explicit MPC. This is because the parallel architecture is very suitable for a super-fast search and cost function computations. But as stated before, explicit MPC memory requirement grows exponentially for larger MPC problem which indeed takes the focus back to parallel computed online MPC.

REFERENCES

[1] J. Rawlings and D. Mayne, "Postface to model predictive control: Theory and design," *Nob Hill Pub*, vol. 5, pp. 155-158, Aug. 2012.

[2] S. J. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control Eng. Pract.*, vol. 11, no. 7, pp. 733–764, 2003.

[3] F. Borrelli, A. Bemporad, M. Fodor, and D. Hrovat, "An MPC/hybrid system approach to traction control," *IEEE Trans. Control Syst. Technol.*, vol. 14, no. 3, pp. 541–552, May 2006.

[4] N. Giorgetti, A. Bemporad, H. E. Tseng, and D. Hrovat, ''Hybrid model predictive control application towards optimal semi-active suspension,'' *Int. J. Control*, vol. 79, no. 5, pp. 521–533, May 2006.

[5] A. Y. Karnik, A. Fuxman, P. Bonkoski, M. Jankovic, and J. Pekar, ''Vehicle powertrain thermal management system using model predictive control,'' *SAE Int. J. Mater. Manuf.*, vol. 9, no. 2, pp. 525–533, Aug. 2016.

[6] A. Gray, Y. Gao, J. K. Hedrick, and F. Borrelli, ''Robust predictive control for semi-autonomous vehicles with an uncertain driver model,'' in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2013, pp. 208–213.

[7] W. Ziyang, Q. Linlin, W. Gang, and L. Xutao, ''Modeling of greenhouse temperature-humid system and model predictive control based on switching system control,'' *Trans. Chin. Soc. Agricult. Eng.*, vol. 7, p. 7, Jul. 2008.

[8] G. Sun and W. Huo, ''Adaptive fuzzy predictive control of satellite attitude based on hierarchical fuzzy systems,'' in *Proc. Int. Conf. Intell. Syst. Design Eng. Appl. (ISDEA)*, Oct. 2010, vol. 1, pp. 208–211.

[9] S. J. Qin and T. A. Badgwell, ''An overview of nonlinear model predictive control applications,'' in *Nonlinear Model Predictive Control*. New York, NY, USA, Springer, 2000, pp. 369–392.

[10] J. A. Nelder and R. Mead, ''A simplex method for function minimization,'' *Comput. J.*, vol. 7, no. 4, pp. 308–313, 1965.

[11] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, ''The explicit linear quadratic regulator for constrained systems,'' *Automatica*, vol. 38, no. 1, pp. 3–20, Jan. 2002.

[12] C. R. Cutler and B. L. Ramaker, ''Dynamic matrix control? A computer control algorithm,'' in *Proc. Joint Autom. Control Conf.*, Sep. 1980, p. 72.

[13] V. Peterka, ''Predictor-based self-tuning control,'' *Automatica*, vol. 20, no. 1, pp. 39–50, Jan. 1984.

[14] N. L. Ricker, ''Model-predictive control: state of the art,'' in *Proc. Chem. Process Control (CPC-IV)*, 199, pp. 271–2961.

[15] K. R. Muske and J. B. Rawlings, ''Linear model predictive control of unstable processes,'' *J. Process Control*, vol. 3, no. 2, pp. 85–96, May 1993.

[16] A. Bemporad and M. Morari, ''Robust model predictive control: A survey,'' in *Robustness Identificat. Control*. New York, NY, USA, Springer, 1999, pp. 207–226.

[17] T. Geyer, F. D. Torrisi, and M. Morari, ''Optimal complexity reduction of polyhedral piecewise affine systems,'' *Automatica*, vol. 44, pp. 1728–1740, Jul. 2008.

[18] F. J. Christophersen, M. N. Zeilinger, C. N. Jones, and M. Morari, ''Controller complexity reduction for piecewise affine systems through safe region elimination,'' in *Proc. 46th IEEE Conf. Decis. Control*, Dec. 2007, pp. 4773–4778.

[19] Y. Nesterov and A. Nemirovskii, *Interior-Point Polynomial Algorithms Convex Programming*, vol. 13, Philadelphia, PA, USA: SIAM, 1994.

[20] T. Glad and H. Jonson, ''A method for state and control constrained linear quadratic control problems,'' *IFAC Proc. Vol.*, vol. 17, no. 2, pp. 1583–1587, Jul. 1984.

[21] D. Axehill and A. Hansson, ''A dual gradient projection quadratic programming algorithm tailored for model predictive control,'' in *Proc. 47th IEEE Conf. Decis. Control*, Dec. 2008, pp. 3057–3064.

[22] S. Richter, C. N. Jones, and M. Morari, ''Real-time input-constrained MPC using fast gradient methods,'' in *Proc. 48h IEEE Conf. Decis. Control*, Dec. 2009, pp. 7387–7393.

[23] D. E. Quevedo, R. P. Aguilera, and T. Geyer, ''Predictive control in power electronics and drives: Basic concepts, theory, and methods,'' in *Advanced and Intelligent Control in Power Electronics and Drives*, New York, NY, USA, Springer, 2014, pp. 181–226.

[24] M. Catucci, J. Clare, and P. Wheeler, ''Predictive control strategies for ZCS direct converter HV power supply,'' in *Proc. Eur. Conf. Power Electron. Appl.*, Sep. 2005, p. 10–pp.

[25] J. Rodriguez, J. Pontt, C. A. Silva, P. Correa, P. Lezana, and P. Cortes, and U. Ammann, ''Predictive current control of a voltage source inverter,'' *IEEE Trans. Ind. Electron.*, vol. 54, no. 1, pp. 495–503, Feb. 2007.

[26] A. Draeger, S. Engell, and H. Ranke, ''Model predictive control using neural networks,'' *IEEE Control Syst.*, vol. 15, no. 5, pp. 61–66, Oct. 1995.

[27] Y. Zhang, ''On the lvi-based primal-dual neural network for solving online linear and quadratic programming problems,'' in *Proc. Amer. Control Conf.*, vol. 2, Jun. 2005, p. 1351.

[28] F. Ke, Z. Li, H. Xiao, and X. Zhang, ''Visual servoing of constrained mobile robots based on model predictive control,'' *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 47, no. 7, pp. 1428–1438, Jul. 2017.

[29] Z. Li, J. Deng, R. Lu, Y. Xu, J. Bai, and C.-Y. Su, ''Trajectory-tracking control of mobile robot systems incorporating neural-dynamic optimized model predictive approach,'' *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 46, no. 6, pp. 740–749, Jun. 2016.

[30] Z. Li, C. Yang, C.-Y. Su, J. Deng, and W. Zhang, ''Vision-based model predictive control for steering of a nonholonomic mobile robot,'' *IEEE Trans. Control Syst. Technol.*, vol. 24, no. 2, pp. 553–564, Mar. 2016.

[31] Z. Li, W. Yuan, Y. Chen, F. Ke, X. Chu, and C. P. Chen, ''Neural-dynamic optimization-based model predictive control for tracking and formation of nonholonomic multirobot systems,'' *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 12, pp. 6113–6122, Dec. 2018.

[32] B. David Kirk and W. Wen-mei Hwu, *Programming Massively Parallel Processors: A Hands-on Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010.

[33] C. Nvidia, ''Programming guide,'' NVIDIA Corp. Org., Santa Clara, CA, USA, Tech. Rep. Version 1.0, 2007.

[34] M. Brdys, M. Grochowski, T. Gminski, K. Konarczak, and M. Drewa, ''Hierarchical predictive control of integrated wastewater treatment systems,'' *Control Eng. Pract.*, vol. 16, no. 6, pp. 751–767, Jun. 2008.

[35] T. Bellemans, B. D. Schutter, and B. D. Moor, ''Model predictive control for ramp metering of motorway traffic: A case study,'' *Control Eng. Pract.*, vol. 14, no. 7, pp. 757–767, 2006.

[36] E. Camponogara, D. Jia, B. H. Krogh, and S. Talukdar, ''Distributed model predictive control,'' *IEEE Control Syst.*, vol. 22, no. 1, pp. 44–52, Feb. 2002.

[37] Y. Wang and S. Boyd, ''Fast model predictive control using online optimization,'' *IEEE Trans. Control Syst. Technol.*, vol. 18, no. 2, pp. 267–278, Mar. 2010.

[38] G. Valencia-Palomo and J. A. Rossiter, ''Efficient suboptimal parametric solutions to predictive control for PLC applications,'' *Control Eng. Pract.*, vol. 19, no. 7, pp. 732–743, Jul. 2011.

[39] I. Nielsen and D. Axehill, ''An o (log n) parallel algorithm for newton step computations with applications to moving horizon estimation,'' in *Proc. Eur. Control Conf. (ECC)*, Jul. 2016, pp. 1630–1636.

[40] D. Soudbakhsh and A. M. Annaswamy, ''Parallelized model predictive control,'' in *Proc. Amer. Control Conf. (ACC)*, Jun. 2013, pp. 1715–1720.

[41] L. Ferranti and T. Keviczky, ''A parallel dual fast gradient method for mpc applications,'' in *Proc. 54th IEEE Conf. Decis. Control (CDC)*, Dec. 2015, pp. 2406–2413.

[42] M. Kögel and R. Findeisen, ''Parallel solution of model predictive control using the alternating direction multiplier method,'' *IFAC Proc. Vol.*, vol. 45, no. 17, pp. 369–374, 2012.

[43] Ion Necoara, Laura Ferranti, and T. Keviczky, ''An adaptive constraint tightening approach to linear model predictive control based on approximation algorithms for optimization,'' *Optim. Control Appl. Methods*, vol. 36, no. 5, pp. 648–666, Sep. 2015.

[44] D. Buchner and P. Skworcow, ''Development and implementation of real-time model predictive control for a three-tank control problem using a dspace multi-processor system,'' *Water Softw. Syst., de Montfort Univ., Gateway, Leicester LE1 9BH, UK*, vol. 36, no. 5, pp. 648–666, Sep. 2015.

[45] M. Brand *et al.*, ''A parallel quadratic programming algorithm for model predictive control,'' *IFAC Proc. Vol.*, vol. 44, no. 1, pp. 1031–1039, 2011.

[46] J. E. Stone, D. Gohara, and G. Shi, ''OpenCL: A parallel programming standard for heterogeneous computing systems,'' *Comput. Sci. Eng.*, vol. 12, no. 3, pp. 66–73, 2010.

[47] A. K. Sampathirao, P. Sopasakis, A. Bemporad, and P. P. Patrinos, ''GPU-accelerated stochastic predictive control of drinking water networks,'' *IEEE Trans. Control Syst. Technol.*, vol. 26, no. 2, pp. 551–562, Mar. 2018.

[48] N. F. Gade-Nielsen, J. B. Jørgensen, and B. Dammann, ''MPC toolbox with GPU accelerated optimization algorithms,'' in *Proc. 10th Eur. Workshop Adv. Control Diagnosis*, 2012, pp. 96–105.

[49] L. Yu, A. Goldsmith, and S. D. Cairano, ''Efficient convex optimization on GPUS for embedded model predictive control,'' in *Proc. General Purpose GPUs*, Feb. 2017, pp. 12–21.

[50] Y. Gang and L. Mingguang, ''Acceleration of MPC using graphic processing unit,'' in *Proc. 2nd Int. Conf. Comput. Sci. Netw. Technol.*, Dec. 2012, pp. 1001–1004.

[51] D.-K. Phung and B. Hérissé, J. Marzat, and S. Bertrand, ''Model predictive control for autonomous navigation using embedded graphics processing unit,'' *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 11883–11888, Aug. 2017.

[52] A. Sadrieh and P. A. Bahri, ''Application of graphic processing unit in model predictive control,'' *Comput. Aided Chem. Eng.*, vol. 29, pp. 492–496, Sep. 2011.

[53] A. Wills, A. Mills, and B. Ninness, ''FPGA implementation of an interior-point solution for linear model predictive control,'' in *Proc. 18th IFAC World Congr.*, 2011, vol. 44. no. 1, pp. 14527–14532.

[54] K. V. Ling, B. F. Wu, and J. M. Maciejowski, ''Embedded model predictive control (MPC) using a FPGA,'' in *Proc. 17th IFAC World Congr.*, 2008, vol. 41. no. 2, pp. 15250–15255.
[55] L. G. Bleris, P. D. Vouzis, M. G. Arnold, and M. V. Kothare, ''A co-processor FPGA platform for the implementation of real-time model predictive control,'' in *Proc. Amer. Control Conf.*, Jun. 2006, pp. 1–6.
[56] S. Longo, E. C. Kerrigan, K. V. Ling, and G. A. Constantinides, ''Parallel move blocking model predictive control,'' in *Proc. Decision Control Eur. Control Conf.*, Dec. 2011, pp. 1239–1244.
[57] J. L. Jerez, G. A. Constantinides, E. C. Kerrigan, and K.-V. Ling, ''Parallel Mpc for real-time FPGA-based implementation,'' *IFAC Proc. Vol.*, vol. 44, no. 1, pp. 1338–1343, 2011.
[58] J. L. Jerez, P. J. Goulart, S. Richter, G. A. Constantinides, E. C. Kerrigan, and M. Morari, ''Embedded online optimization for model predictive control at megahertz rates,'' *IEEE Trans. Autom. Control*, vol. 59, no. 12, pp. 3238–3251, Dec. 2014.
[59] P. Patrinos, A. Guiggiani, and A. Bemporad, ''Fixed-point dual gradient projection for embedded model predictive control,'' in *Proc. Eur. Control Conf. (ECC)*, Jul. 2013, pp. 3602–3607.
[60] H. Peyrl, A. Zanarini, T. Besselmann, J. Liu, and M.-A. Boéchat, ''Parallel implementations of the fast gradient method for high-speed MPC,'' *Control Eng. Pract.*, vol. 33, pp. 22–34, Sep. 2014.
[61] B. Kapernick, S. Suss, E. Schubert, and K. Graichen, ''A synthesis strategy for nonlinear model predictive controller on fpga,'' in *Proc. UKACC Int. Conf. Control (CONTROL)*, Jul. 2014, pp. 662–667.
[62] A. Joos, P. Heritier, C. Huber, and W. Fichter, ''Method for parallel fpga implementation of nonlinear model predictive control,'' *IFAC Proc. Vol.*, vol. 45, no. 1, pp. 73–78, 2012.
[63] A. Joos and W. Fichter, ''Parallel implementation of constrained nonlinear model predictive controller for an FPGA-based onboard flight computer,'' in *Advances in Aerospace Guidance, Navigation and Control*. New York, NY, USA: Springer, 2011, pp. 273–286.
[64] P. D. Vouzis, L. G. Bleris, M. G. Arnold, and M. V. Kothare, ''A system-on-a-chip implementation for embedded real-time model predictive control,'' *IEEE Trans. Control Syst. Technol.*, vol. 17, no. 5, pp. 1006–1017, Sep. 2009.
[65] J. H. Jung and D. P. O'Leary, ''Implementing an interior point method for linear programs on a CPU-GPU system,'' *Electron. Trans. Numer. Anal.*, vol. 28, pp. 174–189, 2008.
[66] E. Smith, J. Gondzio, and J. Hall, ''GPU acceleration of the matrix-free interior point method,'' in *Proc. Int. Conf. Parallel Process. Appl. Math.*. Springer, 2011, pp. 681–689.
[67] J. H. Jung and D. P. O'Leary, ''Cholesky decomposition and linear programming on a gpu,'' Univ. Maryland, 2006.
[68] G. Macindoe, ''Hybrid algorithms for efficient cholesky decomposition matrix inverse using multicore CPUs with GPU Accel,'' Ph.D. dissertation, Dept. Stat. Sci., UCL Univ. College London, U.K., 2013.
[69] X. Meyer, P. Albuquerque, and B. Chopard, ''A multi-GPU implementation and performance model for the standard simplex method,'' in *Proc. 1st Int. Symp. 10th Bal-Kan Conf. Oper. Res.* Thessaloniki, Greece, 2011, pp. 312–319.
[70] M. E. Lalami, V. Boyer, and D. El-Baz, ''Efficient implementation of the simplex method on a CPU-GPU system,'' in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Workshops Phd Forum*, May 2011, pp. 1999–2006.
[71] N. Ploskas and N. Samaras, ''Efficient GPU-based implementations of simplex type algorithms,'' *Appl. Math. Comput.*, vol. 250, pp. 552–570, Jan. 2015.
[72] D. G. Spampinato and A. C. Elstery, ''Linear optimization on modern GPUS,'' in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, May 2009, pp. 1–8.
[73] J. Bieling, P. Peschlow, and P. Martini, ''An efficient GPU implementation of the revised simplex method,'' in *Proc. IEEE Int. Symp. Parallel Distrib. Process., Workshops Phd (IPDPSW)*, May 2010, pp. 1–8.
[74] (2018). *Nvidia Cuda-Cublas Library 2.0*. [Online]. Available: http://developer.nvidia.com/object/cuda.htm
[75] (2018). *Lapack library*. [Online]. Available: http://www.cudatools.com
[76] K. V. Ling, J. Maciejowski, A. Richards, and B. F. Wu, ''Multiplexed model predictive control,'' *Automatica*, vol. 48, no. 2, pp. 396–401, Feb. 2012.
[77] A. Guiggiani, I. Kolmanovsky, P. Patrinos, and A. Bemporad, ''Fixed-point constrained model predictive control of spacecraft attitude,'' in *Proc. Amer. Control Conf. (ACC)*, Jul. 2015, pp. 2317–2322.
[78] G. Knagge, A. Wills, A. Mills, and B. Ninness, ''Asic and FPGA implementation strategies for model predictive control,'' in *Proc. Control Conf. (ECC)*, 2018, pp. 144–149.
[79] F. Xu, H. Chen, X. Gong, and Q. Mei, ''Fast nonlinear model predictive control on FPGA using particle swarm optimization,'' *IEEE Trans. Ind. Electron.*, vol. 63, no. 1, pp. 310–321, Sep. 2016.
[80] D. E. Kirk, *Optimazation Control Theory: Introduction*. Chelmsford, MA, USA: Courier Corporation, 2012.
[81] J. Nocedal and S. J. Wright, *Sequential Quadratic Programming*. New York, NY, USA: Springer, 2006.
[82] M. G. Arnold and J. D. Shuler, ''A synthesis preprocessor that converts implicit style verilog into one-hot designs,'' in *Proc. Meeting Verilog HDL*, Apr. 1997, pp. 38–45.
[83] J. Smoczek and J. Szpytko, ''Particle swarm optimization-based multivariable generalized predictive control for an overhead crane,'' *IEEE/ASME Trans. on Mechatron.*, vol. 22, no. 1, pp. 258–268, Jun. 2017.
[84] C.-H. Wu, S. O. Memik, and S. Mehrotra, ''FPGA implementation of the interior-point algorithm with applications to collision detection,'' in *Proc. 17th IEEE Symp. Field Program. Custom Comput. Mach.*, Apr. 2009, pp. 295–298.
[85] K. Kaczmarski and K. A. Z. P. Rzewski, ''Thrust and CUDA in data intensive algorithms,'' in *New Trends Databases Information Systems*, M. Pechenizkiy and M. Wojciechowski, Eds., Berlin, Germany: Springer, 2014.

KARAM M. ABUGHALIEH received the M.Sc. degree in electrical engineering from Princess Sumaya University for Technology (PSUT), Amman, Jordan, in 2011. He is currently pursuing the Ph.D. degree in electrical and computer engineering with Oakland University, where he is also a Teaching and Research Assistant. He has obtained a very good experience in embedded systems design. He was also involved in object detection and tracking in his master thesis for UAV applications.

SHADI G. ALAWNEH received the B.Eng. degree in computer engineering from the Jordan University of Science and Technology, Irbid, Jordan, in 2008, the M.Eng. and Ph.D. degrees in computer engineering from the Memorial University of Newfoundland, St. John's, NL, Canada, in 2010 and 2014, respectively. Then, he joined the Hardware Acceleration Laboratory, IBM Canada, as a Staff Software Developer, in 2014. After that, he joined C-CORE as a Research Engineer, from 2014 to 2016, and became an Adjunct Professor with the Department of Electrical and Computer Engineering, Memorial University of Newfoundland, in 2016. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Oakland University. He has authored or co-authored scientific publications (including international peer-reviewed journals and conferences). His research interests include parallel and distributed computing, general purpose GPU computing, parallel processing architecture and applications, numerical simulation and modeling, and software design and optimization. He is a Senior Member of the IEEE Computer Society.