2010 IEEE International Conference on Control Applications
Part of 2010 IEEE Multi-Conference on Systems and Control
Yokohama, Japan, September 8-10, 2010

Methods for Efficient Implementation of Model Predictive Control on Multiprocessor Systems

Ruirui Gu, *member IEEE*, Shuvra S. Bhattacharyya, *senior member IEEE* and Williams S. Levine, *IEEE fellow*

Abstract— **Model Predictive Control (MPC) has been used in a wide range of application areas including chemical engineering, food processing, automotive engineering, aerospace, and metallurgy. An important limitation on the application of MPC is the difficulty in completing the necessary computations within the sampling interval. Recent trends in computing hardware towards greatly increased parallelism offer a solution to this problem. This paper describes modeling and analysis tools to facilitate implementing the MPC algorithms on parallel computers, thereby greatly reducing the time needed to complete the calculations. The use of these tools is illustrated by an application to a class of MPC problems.**

I. INTRODUCTION

Model Predictive Control (MPC) has found broad application, especially in the process industry. The main limitation on its application is that it is computationally demanding [1]. As a result, there has been considerable research aimed at speeding up the computation of optimal controls. Most of this research has concentrated on improving the algorithms. Relatively little work [2] has been devoted to improving the implementation of the algorithms.

Recent developments and trends in computing hardware greatly increase the potential for increasing the speed of the MPC computations by properly implementing them in hardware. Specifically, multicore processes are now prevalent. Highly parallel and relatively inexpensive processors, such as the Nvidia GeForce 9800 GX2, with 256 stream processors are available. Because of the inherent tradeoffs between speed and power consumption in computing the current predictions are that this trend will continue, with the number of cores per processor likely to double every two to three years [4]. Further evidence of this trend is that MATLAB now includes a collection of routines for parallel computation.

It can be very time consuming to analyze code line by line in an effort to find ways to implement it on a parallel machine and to minimize the time required for its execution. Furthermore, it can require considerable expertise to do this effectively. Thus, we are developing an analytical and computational framework to assist the user in doing this optimization. The framework utilizes a high level method for modeling control algorithms. The resulting models display the flow of data and the sequencing of calculations in a way that greatly facilitates their analysis. In particular, it is relatively easy to see where computational and/or storage bottlenecks exist. Once identified, these problems can be eliminated or ameliorated by modifying the algorithm or by proper hardware implementation. Furthermore, the approach is hierarchical. It can be applied to components of the algorithm as well as to the overall algorithm. Our work aims to provide a dataflow-based framework to model and analyze computationally intensive control applications and to improve their performance by taking advantage of rapidly developing parallel distributed systems.

In earlier work [5] and [6] we described the basic framework and applied it to develop faster implementations of the Newton-KKT and active set methods for solving quadratic programming problems. The rationale for doing this first was that most MPC problems are solved by the repeated application of one of these two basic procedures. Thus, fast implementations of these algorithms would benefit almost anyone wanting to apply MPC. This paper reports two further developments. The first is straightforward. We have improved the benchmarks for testing our implementations. This is important because better benchmarks result in more accurate estimates of the time needed for the computations. The second improvement is in two parts. We have greatly increased the speed of computation for both Newton-KKT and active set methods by modeling, analyzing, and creating highly parallel implementations of the linear equation solver embedded in both of these algorithms. In order to do this we have had to augment our modeling and analysis tools to include communication delays—an important facet of multiprocessor system performance that should be taken into account carefully when deriving implementations.

II. RELATED WORK

A. Control Background

MPC has been studied at least since the 1970s. At that time various works show an incipient interest in MPC in the process industry [7][8]. The basic ideas appearing in MPC are explicit use of a model to predict the process output at future time instants; calculation of a control sequence minimizing a certain objective function; and the application

R. Gu is with the Department of Electrical and Computer Engineering, and Institute for Advanced Computer Studies, University of Maryland, College Park, MD, 20742. rgu@umd.edu

S. S. Bhattacharyya is with the Department of Electrical and Computer Engineering, and Institute for Advanced Computer Studies, University of Maryland, College Park, MD, 20742. ssb@umd.edu

W. S. Levine is with the Department of Electrical and Computer Engineering, University of Maryland, College Park, MD, 20742. wsl@umd.edu

of only the first control signal of the sequence calculated at each step. A detailed introduction to MPC and some specific algorithms can be found in the book [9].

It is well known that MPC can be computation intensive and that, as a result, it can usually be used only in applications with relatively slow dynamics [1]. One would like to be able to compute the controls in real time by solving an optimal control problem. This has prompted a number of researchers to investigate means for increasing the speed with which optimal controls can be computed. Much of this work has focused on improving the algorithms [1], [11].

A few researchers have addressed the implementation of MPC. Ling et al. [12] demonstrated that a "reasonably sized constrained MPC Controller" could be implemented on a modest FPGA chip. Bleris et al. [13] have proposed a computing architecture that is specifically designed for MPC. Furthermore, they have proposed a design framework for application specific processor implementation [2]. Our approach differs from that of Bleris et al. in that we focus on modeling the MPC algorithm structure. This model can be used to derive efficient implementations across a range of architectures. In particular, designers can systematically trade off performance and resource requirements, based on the constraints of the control problem, and the set of available hardware resources.

B. Embedded Signal Processing Background

Since the mid 1980s, a class of graphical program representations has been evolving steadily, and gaining increasing acceptance among designers of digital signal processing (DSP) systems. Synchronous dataflow (SDF) is a specialized form of dataflow that is streamlined for efficient representation of DSP systems [17]. Since the introduction of SDF, a variety of such DSP-oriented dataflow models of computation have been proposed, and DSP-oriented models have been incorporated into many commercial design tools, including Agilent ADS, Cadence SPW (later acquired by CoWare), National Instruments LabVIEW, and Synopsys CoCentric. These alternative modeling approaches provide different trade-offs among expressive power (the range of DSP applications that can be represented), analysis potential (the rigor with which implementations can be automatically validated or optimized), and intuitive appeal.

A limitation of SDF and related models, such as cyclo-static [18] dataflow, is that dynamic dataflow relationships among computations cannot be described. Earlier work on DSP-oriented dataflow models has focused heavily on static dataflow techniques, especially SDF. As designers seek to develop more and more complex embedded DSP systems, incorporating more flexible sets of features, and more powerful forms of adaptivity, exploration of dynamic dataflow models is becoming increasingly important. In this paper, we describe a new dynamic dataflow modeling technique, called reactive, control-integrated dataflow (RCDF), that appears particularly promising for MPC applications. Our approach is more specialized compared to other dynamic dataflow techniques, but for MPC, this specialization can be exploited in useful ways to streamline the implementation process.

Note that in addition to their formal properties, DSP-oriented dataflow models provide different kinds of software architectures for working with signal processing computations (of which control system implementations form an important sub-class). This kind of representation can help to structure subsequent phases of design, simulation, verification, testing, and implementation *regardless of whether the underlying model of computation is explicitly supported by an off-the-shelf design tool*. This is true especially in the area of embedded systems, including embedded control, where designers are often willing to explore specialized, application/architecture-driven analysis techniques that may provide streamlined performance, power consumption, cost, or robustness.

III. DATAFLOW BASED FRAMEWORK FOR MODEL PREDICTIVE CONTROL

To facilitate efficient implementation of MPC applications, we have introduced a form of dataflow called Reactive Control integrated Dataflow (RCDF), which provides a way to model reactive control structures that are relevant to MPC computations [5]. Reactive Control integrated Dataflow (RCDF) is an extension of SDF, which introduces a way to model reactive control structures. The RCDF model provides a set of mutually-exclusive edges (MEs) and imposes restrictions on the number of tokens produced or consumed on the edge when the source or sink actor, respectively, of the edge executes. Among the MEs, two kinds of special MEs *mutually-exclusive token production edges* (MTPE), and *mutually-exclusive token consumption edges* (MTCE) are especially useful when modeling different reactive control structures such as switch and reset.

In practice, many MPC problems involve repeated solutions of:

$$\text{minimize} \sum_{k=0}^{N-1}(x^{'}(k)C^TCx(k)+u^{'}(k)u(k))+x^{'}(N)C^TCx(N)$$

s.t.

$$x(k+1) = Ax(k) + Bu(k), k = 0, \cdots, N-1$$

$$|u_i(k)| \leq u_{max}, i = 1, \cdots, m, k = 0, \cdots, N-1$$

$$x(0) = x_0, x_0 \text{ is constant}$$

Here A is an $n \times n$ matrix, B is an $n \times m$ matrix, and C is a $p \times n$ matrix. $x(k)$ is a $n \times 1$ vector, and $u(k)$ is an $m \times 1$ vector. For simplicity of notation it has been assumed that all the controls are weighted equally. This assumption can be trivially relaxed.

In order to create a family of benchnark problems to use in evaluating and testing our implementations of MPC, we randomly chose 50 values for the three matrices A, B, and C, all sets with $n = 10$, $m = 8$, and $p = 8$. We then checked whether (A, B) was controllable. If not, we deleted that trio A, B and C from the set. If they were controllable we then checked if (A, C) was observable. If not, then we deleted that A, B and C. The remaining trios of matrices constitute a collection of test problems of randomly varying computational difficulty. To complete the problem formulation, we chose $N = 50$.

In order to reduce the resulting MPC problems to a form in which the Newton-KKT or active set methods can be easily applied, we formed the large matrices given below:

$$\hat{A} = \begin{bmatrix} B & 0 & \cdots & 0 \\ AB & B & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ A^{N-1}B & A^{N-2}B & \cdots & B \end{bmatrix},$$

$$\hat{C} = \begin{bmatrix} C^{'}C & 0 & \cdots & 0 \\ 0 & C^{'}C & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & C^{'}C \end{bmatrix}, \hat{d} = \begin{bmatrix} A \\ A^2 \\ \cdots \\ A^N \end{bmatrix} * x_0,$$

The result is the quadratic programming problem $< P >$:

$$\text{minimize } (\hat{A}\hat{u} + \hat{d})^T \hat{C} (\hat{A}\hat{u} + \hat{d}) + \hat{u}^T \hat{u}$$

subject to

$$|u_i(k)| \leq u_{max}, i = 1, \cdots, m, k = 0, \cdots, N-1$$

where $\hat{u} = [u^T(0)\ u^T(1) \ldots u^T(N-1)]^T$, and $(.)^T$ denotes transpose. Note that each of the $u(k)$ is an m-vector so the overall dimensions of $\hat{u}$ are $Nm \times 1$.

In previous work [5], [6], we modeled, analyzed, and improved the implementation of both the Newton KKT and active set methods for solving the general QP problem. As a first test we used the optimized versions of the Newton-KKT and active set methods to solve our new benchmark problems. The results are shown in Table I.

The simulation results were obtained using a desktop computer with a 1.30GHz processor. seq denotes sequential implementation of the Newton KKT algorithm; $Newton-KKT_p$ is our improved implementation of the Newton KKT algorithm; and $active{-}set_P$ is our improved implementation of the active set method.

IV. Newton KKT Incorporating a Parallel Linear System Solver

A. Newton KKT

Figure 1 illustrates a model, developed in [5], of the Newton KKT algorithm based on RCDF. We implemented communication between actors based on the dataflow model, however implementation of each actor used purely sequential

TABLE I

Simulation results for MPC problems: execution time for different scenarios (sec).

statistics	mean	variance
seq	0.52656	0.053714
$newton - KKT_p$	0.16953	0.0041189
$active - set_p$	0.18098	0.0012329

programming. As shown in Figure 1, there are seven actors in the system, and each actor is responsible for a certain function. The function of each actor is described in brief as follows:

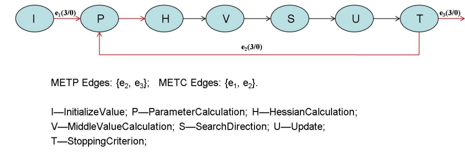

Fig. 1. RCDF model of Newton KKT algorithm. $\{e_2, e_3\}$is a set of METP Edges, and $\{e_1, e_2\}$ is a set of METC Edges.

I—initializes the values of state variables and the values of the parameters, which are used later such as tolerance threshold.

P—computes the values of f, g and the Schur component at the current value of x for every iteration.

H—computes the modified Hessian matrix. It functions when the Hessian matrix has some eigenvalues equal to 0.

V—computes the gradient of f in every iteration.

S—computes the search direction for the next iteration. It finds the solution by solving a linear system of equations. This is where Newton's method is used.

U—computes the updated values of x, f and g.

T—compares the difference between the updated value and the previous value with a given criterion, to see if the system needs to go to the next iteration or terminate in this iteration.

Since the actors are divided based on functionality, code size is different from one actor to another. The simplest actor may be composed of only one addition. A much more complex actor may be expanded as a dataflow subgraph, such as that represented by the hierarchical actor U in Figure 1.

We conducted simulations in MATLAB to evaluate the time each actor consumes. From the profiling result in Table II, it is obvious that H, S and U are computation intensive actors compared with actor V.

In our previous work, we applied functional parallelism to the actors H and S. The modified RCDF model is shown in Figure 2.

We carefully transformed actor U to derive an efficient implementation for it, as shown in Figure 3. Since the original dataflow model was based on a sequential programming language, this is a sequential model in terms of computation.

We transformed the dataflow model in order to make use

TABLE II

EXECUTION TIME IN SECONDS FOR DIFFERENT ACTORS IN NEWTON KKT. $ncond$ IS THE PARAMETER TO GENERATE DIFFERENT QP PROBLEMS.

$ncond$	0		3		6	
statistics	mean	var	mean	var	mean	var
H	0.0122	4e-5	0.0108	5e-5	0.0125	4e-5
S	0.0082	6e-5	0.0056	5e-5	0.0259	7e-5
U	0.0048	5e-5	0.0049	5e-5	0.0102	5e-5
V	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

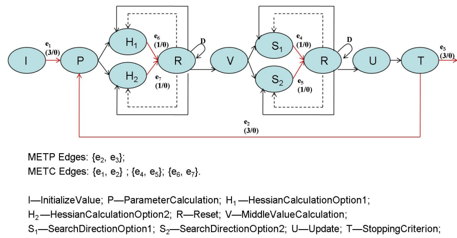

Fig. 2. Modified RCDF model of Newton KKT algorithm: modified actor H and actor S. H_1−HessianCalculationOption1; H_2−HessianCalculationOption2; R−MVOS; S_1−SearchDirectionOption1; S_2−SearchDirectionOption2.

of parallelism. The transformed dataflow model is shown as Figure 4. In the transformed RCDF model, actors U_{t1} and actor U_{t2} contain independent computations for a set of data. If these computations are implemented in a multi-processor system, we can improve system performance.

B. Parallel Linear System Solver

From Table II, we can see S is one of the computational bottlenecks. The main computational part of S is to solve a linear system of equations. Similar linear systems of equations play an important role in many problems in control

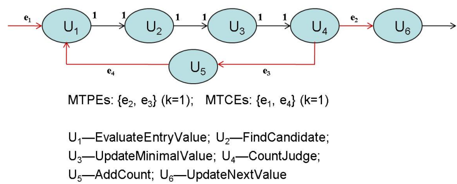

Fig. 3. Sequential RCDF model of actor U

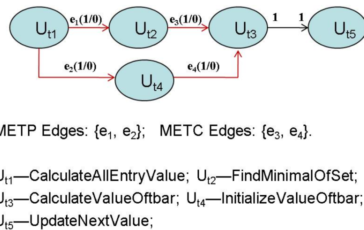

Fig. 4. Modified RCDF model of actor U

and signal processing. Because of the great importance of solving linear equations in science and engineering there is a vast literature on the parallel computation of the solution to such problems. This literature is both complex and confusing because parallel computing can be very sensitive to the details of the computer architecture as well as to the algorithm used. Furthermore, because solving such problems is one of the benchmarks for determining the fastest computer, programmers have considerable incentive to develop special tricks to make specific computers solve such problems quickly.

However, it is clear from the literature that very large improvement in the speed with which linear equations are solved is possible using various forms of parallel computing. Furthermore, there is a large variety of parallel hardware and this collection is rapidly increasing. In order to take advantage of this we have first enhanced RCDF to include a way to account for communication delays because such delays are very significant in highly parallel computing. We have also begun to explore ways to implement large amounts of parallelism in the linear equations solver that is a major component of both Newton-KKT and active set methods for solving QPs. This work is described below.

Gaussian Elimination (GE) is a general way to solve linear systems of equations and its parallel implementation has been heavily studied. Thus, although the QR method is arguably better for the class of problems of interest here, it is better to begin with GE. Implementations of parallel Gaussian Elimination depend on the parallel hardware platform, such as a multiprocessor or multicore system. Computations in each processing unit are similar to each other. However execution of the computations requires the collaboration of all the units.

The problem we wish to solve has the form $Ax = b$. Note that the A and b here are completely different from the A and B in the MPC problems. Here A is simply a square invertible matrix and b is a vector of commensurate size.

Grid computation, such as in ScaLAPACK, is typical for parallel Gaussian Elimination. In this way, key computations are identified and then distributed in a processor matrix.

Figure 5 presents our RCDF model of one processing unit for Gaussian Elimination.

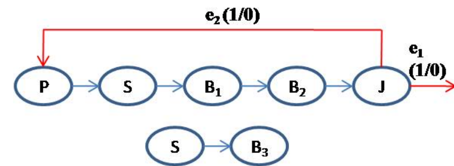

Fig. 5. RCDF model of Gaussian Elimination on Single Processing Unit. The actors are as follows: S-swap, P-findPivotRow, Bn-BLASn, J-judge. $e1, e2$ is a set of MTPEs.

In the RCDF model, there is a particularly important set of actors, indicated by BLASn (Basic Linear Algebra Subprograms). BLASn represents a library to perform basic

linear algebra operations such as vector and matrix multiplication [?]. The BLAS are used to build larger packages such as LAPACK. Because they are heavily used in high-performance computing, highly optimized implementations of the BLAS have been developed by hardware vendors such as Intel and AMD. The LINPACK benchmark relies heavily on DGEMM, a BLAS subroutine, for its performance.

Parallel Gaussian Elimination requires the collaboration of multiple processing units. Although each processing unit conducts similar computation tasks, they have to communicate with each other to swap the data and calculate the final result. An RCDF model to implement Gaussian Elimination on 4 processors is shown in Figure 6. In this model, the processor matrix for GE is 2x2. Note that communication edges have been introduced to indicate the communication between two actors. This is different from other types of edges in RCDF models; there is no specific token related to communication edges.

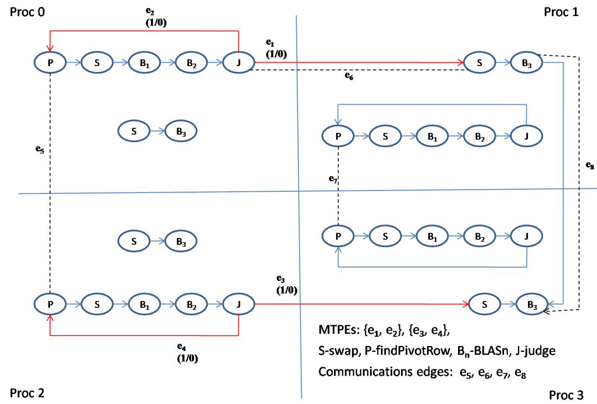

Fig. 6. RCDF model of Gaussian Elimination on Four Processing Units

In our target architecture model, we map processors onto a 2-dimensional matrix in a block-cyclic distributed manner. Such an arrangement is represented in the form $n_r \times n_c$, where n_r represents the number of processors in a row, and n_c represents the number of processors in a column of the target architecture matrix. The matrix to be processed is also divided into a 2-D pattern, based on homogeneous blocks of size $m_r \times m_c$, where $m_r \leq n_r$, and $m_c \leq n_c$. In our experiments, it is assumed that $m_r = m_c$ (i.e., each block in the pattern has a "square" arrangement). The computations related to blocks are allocated to the processor pattern in modulus fashion — after a computation is mapped to the last row or column, the mapping process "wraps around" cyclically to the first row or column, respectively.

We simulate our model of a distributed memory environment using the Message Passing Interface (MPI). MPI is commonly used to simulate the communications between different processing units in a system with distributed memory. One of the major aspects of implementing the Gaussian Elimination algorithm on a distributed memory system is that the communication time has to be taken into account when calculating the total execution time. In general, inter-processor communication time has a significant effect on the performance of algorithms on multiprocessor systems.

TABLE III

SIMULATION RESULTS FOR PARALLEL GAUSSIAN ELIMINATION WITH DIFFERENT PROCESSOR PATTERNS (SEC).

Process pattern	mean	variance
2x2	2.000129	0.102159
2x4	1.035022	0.011020
2x8	0.795640	0.072004
2x16	0.581850	0.025809
2x2	1.985206	0.100000
4x2	1.029348	0.021832
8x2	0.808240	0.052389
16x2	0.562020	0.013480

In our experiments, we use a constant time of 0.002sec as the communication overhead between any two processors. The overhead is chosen to be comparable to the computation time for reasonable simulation. Whenever there are communication between two processors, as indicated by the communication edges shown in Figure 6, the communication overhead estimate is added onto the total execution time.

By applying the Schur complement to problem $< P >$ [22], we decreased the dimensions of the linear system from 1200x1200 to 400x400. We tested the Gaussian Elimination algorithm with different processor patterns given the fixed block size to be allocated in each processor. The $PDGESV$ routine in ScaLAPACK is used in the simulation. The simulation results are shown in Table III. The numbers in the table were determined in the following way. The benchmark QPs set up earlier involving $\hat{A}$, $\hat{C}$, $\hat{d}$ with $\hat{u}$ as the unknown to be computed were input to our improved implementation of Newton-KKT. This created a large system of linear equations to solve. This system has some structure which we exploited to simplify the computations slightly. Almost all of this special structure is always present in QPs derived from an MPC problem. We then applied parallel GE in the various ways indicated in Table III to obtain the indicated results.

The simulation results indicate the effect of communication time between processors. The system performance does improve with an increasing number of processors, however, the rate of increase decreases as the number of processors used in the computation increases. The reason is that it takes time for the processors to communicate and synchronize with each other. The portion of communication time in the total execution time increases with the increasing number of processors.

In our simulations, we assume that all the processors are homogenous, which means that each processor has the same capacity of computation. Under this assumption, the processor pattern 2×4 results in the same speed as the pattern 4×2. The results will change if we apply heterogeneous

TABLE IV

SIMULATION RESULTS FOR PARALLEL GAUSSIAN ELIMINATION WITH DIFFERENT BLOCK SIZE (SEC).

Block size	mean	variance
5	1.193409	0.032600
10	0.906433	0.052802
20	0.795640	0.072004
30	0.606800	0.012560
40	0.523929	0.021430
80	0.752324	0.014398

TABLE V

SIMULATION RESULTS FOR MPC PROBLEMS WITH PARALLEL GAUSSIAN ELIMINATION (SEC).

statistics	mean	variance
seq	10.3689	1.309000
newton-KKT	6.27500	0.004994
pges	7.36600	0.247642
pgen	3.92840	0.024239

processors.

We also tested the parallel Gaussian Elimination algorithm with the same processor pattern but different block sizes. The same matrices were used as in the tests that determined the values in Table III. The simulation results are shown in Table IV.

Simulation results indicate that the system performance achieves its peak when the block size is 40. This is also the effect of communications between different processors. In the extreme case, if we allocate the computation of each entry of the matrix into its own processor, the communication time will dominate the execution time. On the contrary, if we allocate the whole matrix as one block, it turns out to be a sequential Gaussian Elimination instead of a parallel version.

If we integrate the parallel Gaussian Elimination with the 2×8 pattern into the Newton-KKT system, the simulation results are shown in Table V. In Table V, seq denotes a sequential implementation of MPC problems using the standard Newton-KKT method; $newton - KKT_s$ denotes parallel implementation without a parallel linear solver; $pges$ is sequential implementation with only the linear system solver executed in a parallel way; $pgen$ is a parallel implementation with parallel Gaussian Elimination.

The performance of Newton-KKT with parallel Gaussian elimination will be further improved if we use more processors than 16. However, with 2×8 processors, the actor S is no longer the computational bottleneck; it is not necessary to consume more hardware resources.

V. CONCLUSIONS

In this paper, we have proposed a general framework for modeling, analyzing, and developing fast parallel implementations of the algorithms used in MPC. We have illustrated the use of this approach by application to the Newton-KKT part of the computations for a practically important class of MPC problems. We have demonstrated in simulations that this approach does result in implementations of MPC that require much less computing time.

Much remains to be done. One example is that the QR algorithm is a better candidate for solving the system of linear equations within Newton-KKT and active set methods. Parallel implementations of the QR algorithm need to be developed. Furthermore, because the communication times are greatly dependent upon the specific hardware the methods described here need to be applied to the different examples of hardware.

The full collection of MPC algorithms is much richer than those analyzed here. Many MPC algorithms are much more complicated and require much more time than the ones analyzed here. These techniques have the potential to greatly decrease the time needed to solve these more complicated MPC problems. Lastly, there is considerable opportunity to improve on the benchmarks we have developed.

REFERENCES

[1] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," in *Proceedings of the IFAC World Congress*, July 2008, pp. 6974–6979.

[2] L. G. Bleris, J. Garcia, M. G. Arnold, and M. V. Kothare, "Towards embedded model predictive control for system-on-a-chip applications," *Journal of Process Control*, vol. 16, no. 3, March 2006.

[3] Y. K. Chen, C. Chakrabarti, S. S. Bhattacharyya, and B. Bougard, "Signal processing on platforms with multiple cores: Part 1–overview and methodologies," *IEEE Signal Processing Magazine*, vol. 26, no. 6, pp. 24–25, November 2009.

[4] R. Gu, S. S. Bhattacharyya, and W. S. Levine, "Dataflow-based implementation of model-predictive control," in *Proceedings of American Control Conference*, June 2009, pp. 2343–2349.

[5] ——, "Improving the performance of active set based model predictive controls by dataflow methods," in *Proceedings of the 48th IEEE Conference on Decision and Control held jointly with the 2009 28th Chinese Control Conference*, December 2009, pp. 339–344.

[6] J. Richalet, A. Rault, J. L. Testud, and J. Papon, "Model predictive heuristic control: application to industrial processes," *Automatica*, vol. 14, no. 2, pp. 413–428, 1978.

[7] C. R. Cutler and B. Ramaker, "Dynamic matrix control—a computer control algorithm," in *Proceedings of the Joint Automatic Control Conference*, 1980.

[8] E. F. Camacho and C. Bordons, *Model predictive control in the process industry*. Springer, 1995.

[9] H. Chung, E. Polak, and S. Sastry, "An accelerator for packages solving discrete-time optimal control problems," in *Proceedings of the IFAC World Congress*, July 2008, pp. 14 295–14 300.

[10] K. V. Ling, S. P. Yue, and J. M. Maciejowski, "An FPGA implementation of model predictive control," in *Proceedings of the American Control Conference*, June 2006.

[11] 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 *Proceedings of the American Control Conference*, June 2006.

[12] E. A. Lee and D. G. Messerschmitt, "Synchronous dataflow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, September 1987.

[13] G. Bilsen, M. Engels, R. Lauwereins, and J. A. Peperstraete, "Cyclo-static dataflow," *IEEE Transactions on Signal Processing*, vol. 44, no. 2, pp. 397–408, February 1996.

[14] P. A. Absil and A. L. Tits, "Newton-kkt interior-point methods for indefinite quadratic programming," *Computational Optimization and Applications*, vol. 36, no. 1, pp. 5–41, January 2007.