Efficient Implementation of Particle Swarm Optimization Algorithm

Ruirui Gu, Zhefu Shi
R. Gu is with Microsoft, One Microsoft Way Redmond, WA, 98052, ruiruigu@microsoft.com
Z. Shi is with University of Missouri – Kansas City, zhefushi@mail.umkc.edu

Abstract

Dataflow representations have been developing since the 1980's. They have proven to be useful in identifying bottlenecks in DSP algorithms, improving the efficiency of the computations, and in designing appropriate hardware for implementing the algorithms. This paper extends and demonstrates the use of dataflow-based methodology, called as Reactive Control-integrated Dataflow based Aggressive Forwarding Check (RCDF-AFC), to implement a Particle Swarm Optimization (PSO) algorithm in hardware-software co-design. PSO has been extensively used in the real world but its application has been limited to relatively slow processes because it is computationally intensive. It is shown that the time required for the PSO computations using extensive iteration for solving the optimization problem can be reduced by means of dataflow based analysis and implementation improvements based on these analyses.

Keywords: *Particle swarm optimization (PSO), Model Predictive Control (MPC)*

1. Introduction

Model Predictive Control (MPC) provides an elegant framework for higher accuracy. In previous research [1], we used dataflow techniques to improve the speed of MPC algorithms that utilize the Newton-KKT method or the active set method to solve for the control value. In this paper, we extend our work by applying similar but different dataflow techniques to improve the
performance of another nonlinear control algorithm, particle swarm optimization (PSO), which can be used in the nonlinear MPC controllers [2].

The motivation to the work is based on two facts. One fact is although MPC has been applied to many practical problems, it is computationally intensive. Due to intensive computation takes a long time interval, MPC does not fit real time applications although it can provide more accurate results. The second fact is that hardware-software design makes it possible to efficient implement traditional algorithms in parallel hardware platforms, such as fast development of the homogeneous multi-core parallel processors.

As a result, there has been considerable research aimed at speeding up the computation of optimal controls for MPC. Most of this research has concentrated on improving the algorithms. Some work [3] has been devoted to improving the implementation of the algorithms. It is well known that an algorithm that can be executed with many parallel steps will be much faster if properly implemented than a more efficient algorithm that must operate sequentially. However it is not practical to assume availability of unlimited parallel processing units.

The paper is organized as follows. We generalize the related work in the next section. We describe the dataflow framework to be applied to PSO in the following section. We then elaborate the method of Reactive Control-integrated Dataflow based Aggressive Forwarding Check (RCDF-AFC), which can be used to improve the MPC system performance. Finally, we conduct the simulation in Matlab and compare the results with the function provided by Matlab library.

2. Related work

Some related work in both controls and computer engineering was summarized in our earlier papers [1][2]. In this section, we focus on efficient implementation of computational intensive algorithms on parallel systems.

There are already some works on implementation of complicated control systems, such as MPC. [4] proposed a generic solver to specify Receding Horizon Control (RHC) policy using Matlab. [5] proposed a generalized C++ class to solve nonlinear MPC and dynamic optimization problems using BzzMath Library. However, none of the work above has exploited the usage of parallel systems explicitly.

There are also some works to introduce parallel implementation into modern control fields. One example is parallel version of Linpack [6]. Our previous work also started the exploration. However, there is little work on how to make it practical by taking into consideration of hardware limits, as well as how to use parallel processing unit more efficiently. The hardware constraints come from two aspects: on one hand, the number of parallel processing units is limited. It is not only from constraints of hardware manufacturing, but also from the fact that the energy consumption increases dramatically with increasing number of parallel processing units. On the other hand, load balance and scheduling are important for parallel systems. Without careful tuning the software on hardware platforms, it is of high probability that the hardware platform with more processing units performs worse than the one with less processing units, considering more overhead of time on communications between processing units introduced with more processing units. Our work addressed the issue to implement complicated control algorithms while taking into consideration of hardware-software codesign, which is promising direction to a complete and practical solution.

3. PSO and the RCDF model

3.1. PSO

Particle swarm optimization (PSO) is a population based stochastic optimization technique developed by Eberhart and Kennedy in 1995 [7], inspired by social behavior of bird flocking or fish schooling. PSO algorithm is especially useful for complicated non-linear optimization problems. In PSO, the potential solutions, called particles, come from the problem space by following the current optimum particles. PSO optimizes a problem by having a population of particles, and moving these particles around in the search-space according to simple mathematical formulae over the particle’s position and velocity. The advantages of PSO are that PSO is easy to implement and there are fewer parameters to adjust. PSO has been successfully applied in many areas: function optimization, artificial neural network training, fuzzy system control, etc. PSO is a good candidate as an optimization technique to solve the constrained nonlinear optimization problem in a MPC model.

PSO is initialized with a group of random particles (solutions) and then searches for the optimal objective value through calculation iterations. In every iteration, each particle is updated by following two “best” values. The first one is the best solution this particle has achieved so far, which is a local optimal value and is called *lbest*. Another “best” value which is tracked by the particle swarm optimizer is the best value, obtained so far by all particles. This best value is a global optimal value and is called *gbest*.

After finding the two best values, the particle updates its velocity and positions with following equation (1) and (2).

$$v[] = v[] + c1 * rand() * (pbest[] - present[]) + c2 * rand() * (gbest[] - present[]) \tag{1}$$
$$present[] = persent[] + v[] \tag{2}$$

where *v*[] is the particle velocity; *persent*[] is the current particle position (solution); *pbest*[] and *gbest*[] are defined as stated before; *rand*() is a random number between (0,1); *c1* and *c2* are learning factors.

In this paper, for nonlinear MPC systems, we applied the dynamic dataflow modeling technique, called Reactive Control-integrated Dataflow (RCDF) [1] to PSO optimization. This approach is more specialized than other dynamic dataflow techniques, but for PSO and other control algorithms, this specialization can be exploited in useful ways to streamline the implementation process.

We need very little of the full RCDF formalism. It is mainly necessary to understand that for DSP-oriented dataflow graphs, vertices (actors) represent computations of arbitrary complexity, and an edge represents the flow of data as values are passed from the output of one computation to the input of another. Each data value is encapsulated in an object called a token as it is passed across an edge. Actors are assumed to execute iteratively, over and over again, as the graph processes data from one or more data streams. These data streams are typically assumed to be of unbounded length (e.g., derived implementations are not dependent on any predefined duration for the input signals). In dataflow graphs, interfaces to input data streams are typically represented as source actors (actors that have no input edges).

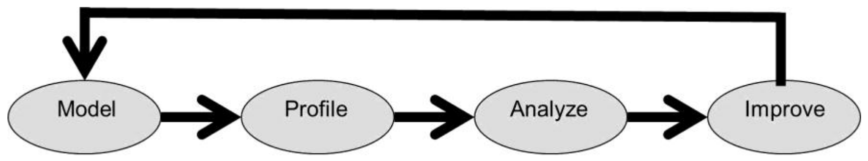
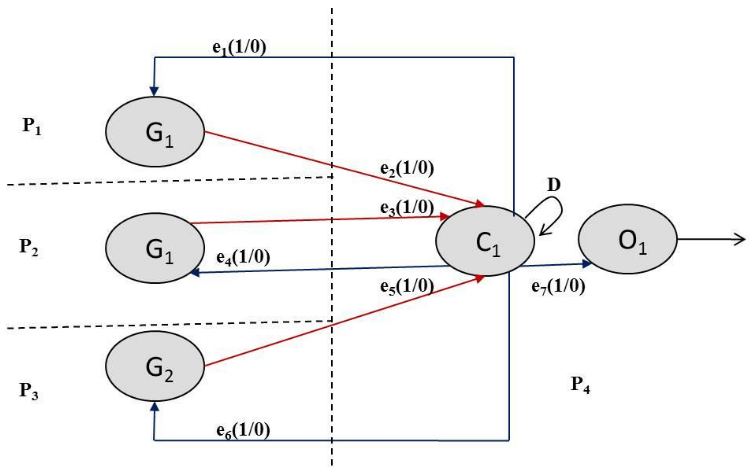


Figure 1. Dataflow Framework for efficient system implementation.

3.2. Applying RCDF model to PSO on a homogeneous multicore system

The dataflow framework provides a complete solution from system modeling to optimized implementation, as shown in Figure 1. First of all, the control algorithm is modeled as a dataflow model. After all the computation tasks are divided into different actors, we profile the execution time of each actor to determine the bottleneck(s) of the system performance.



METP Edges: $\{e_1, e_4, e_6, e_7\}$; METC Edges: $\{e_2, e_3, e_5\}$

G_i—Instance of the actor G to compute the objective function; (i = 1, 2, 3)

C_1— Instance of the actor C to compare the results from each seed;

O_1— Instance of the actor O to output the result;

Figure 2. The RCDF model of the Particle Swarm Optimization implementation.

3.3. Dataflow model of PSO

In this paper, we applied the RCDF model to PSO algorithm, as shown in Figure 2. As an important input to hardware-software design problem, the number of processing units is limited, and we assume there are four processing units.

The functionality of actors is described as follows:

G—The actor G is used to compute the objective function based on the given feed. G can be a super actor since the computation inside this actor can be extended into a sub dataflow graph. The further exploration is beyond the scope of this paper. The instance of this actor produces the result based on the given seed as token and then sends it to the instance of actor C.

C—The actor C is used to compare the results from different processing units in each iteration. The instance of this actor is also responsible for sending seeds to available processing unit, as well as sending the result to the instance of actor O, depending on stopping criterion and iteration number.

O—The actor O is used to collect the result in the current subspace. Based on our dataflow-based modeling approach, along with MATLAB implementations of the individual actors, we have conducted MATLAB simulations to evaluate the contribution of each actor to the overall execution time required for the application. We do not elaborate profiling phase due to limit of the paper size. In this specific model, the computation time required from the instance of the actor G is much longer than other actors. In other words, the actor G is dominant in terms of computation time. Given limited number of processing units, we instantiate multiple instances of actor G in order to take advantage of parallel systems. The instance of actor C and the instance of the actor O are put in the same processing unit.

The METC, as shown in Figure 2, consists of a set of edges with the destination of C. C consumes only one token from one of the instances of the actor G at one time. The METP, as shown in Figure 2, consists of a set of edges from the source of C. At one time, C produces only one token for either one of the instances of the actor G_i or O_1. The computation of G_i and O_1 is relative light and can be negligible when they are hosted in P_4.

Each instance of G_i is hosted in one processing unit. Once it finishes computing the result of the seed for the current iteration *k*, it will send the result back to C_1. Once C_1 receives the result from the instance G_i, e.g. G_1, it knows that the processing unit P_1 is available now. First it will check to see if there is any seed untouched in the current iteration. If there is, C_1 will send the next seed in the sequence to P_1. If no and if the maximum iteration is reached, C_1 will send the final result to O_1; if the maximal iteration is not reached, C_1 will start a new iteration *k*+1 and then send the first seed S_{lk+1} for *k*+1 to P_1. The methodology to choose S_{lk+1} is discussed in details in next section.

4. AGGRESSIVE FORWARDING CHECK

In this section, we propose the RCDF based aggressive forwarding check (RCDF-AFC) algorithm, and describe the performance gain from the algorithm.

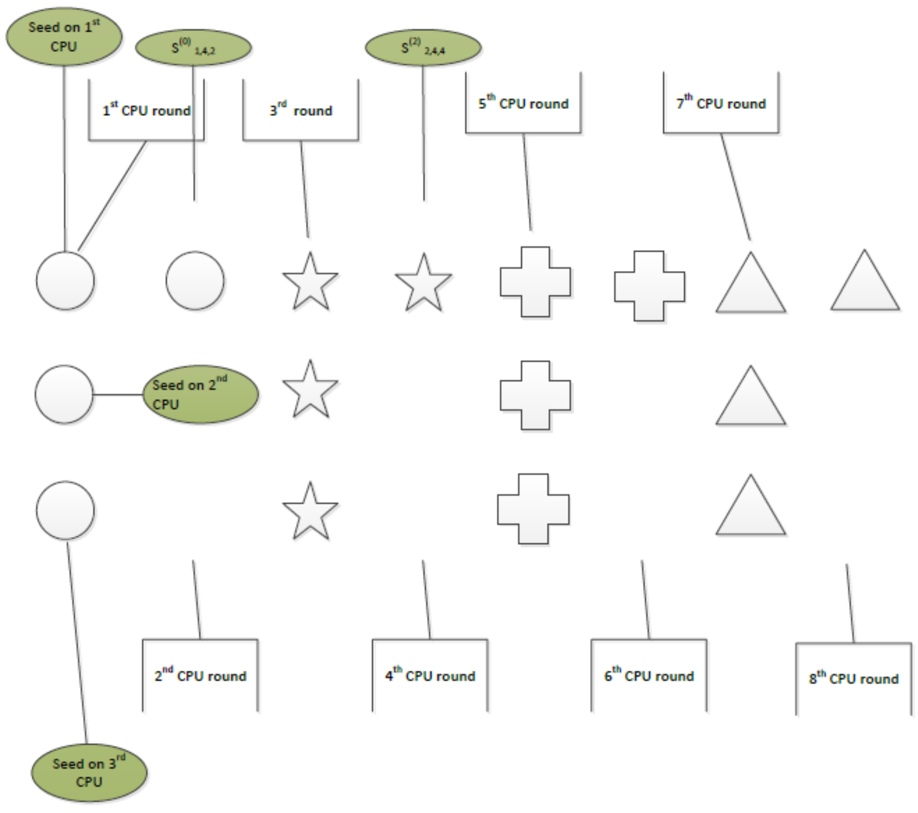


Figure 3. Workflow without RCDF-AFC: 4 seeds running on 4 CPUs.

We conduct our research on homogeneous multicore system, so the computation time of each CPU on single seed is identical. Suppose we have X seeds in each iteration of PSO algorithm, and the number of CPUs available is Y. Among these CPUs, one CPU is responsible for C_1 in Figure 2, and the remaining Y - 1 CPUs are responsible for the instances of G. In this paper, we assume $Y - 1 < X$, which means the number of CPUs is limited and one iteration of feeds computation cannot be finished in one CPU round.

Fig. 3 shows the case that 4 seeds are running on 4 CPUs without RCDF-AFC. The CPU acting as C_1 is considered as monitoring the workflow and keeps the optimized value for the next iteration, and it is not explicitly shown on Fig. 3.

Since the initial seed of the second iteration is determined by the best value of the results from all the seeds in the first iteration, it requires 2 CPU rounds to finish the first iteration, although all other CPUs will be idle when the fourth seed is using only one CPU in the second CPU round. As shown in Fig. 3, two of three CPUs are always idle during the second round of each iteration.

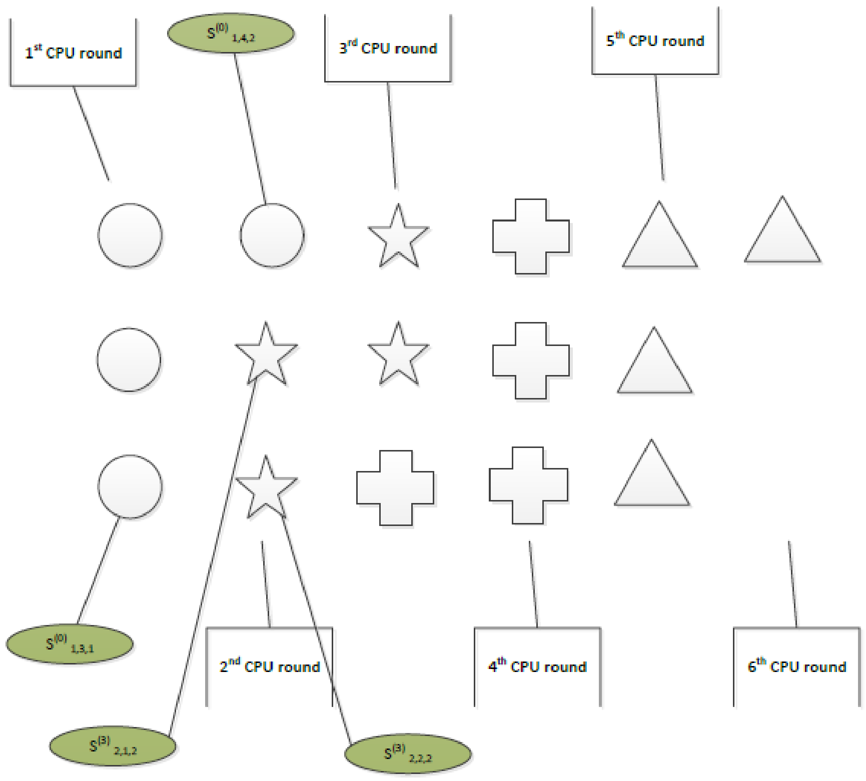


Figure 4. Work flow with RCDF-AFC: 4 seeds running on 4 CPUs.

We apply RCDF-AFC to the process, as shown in Fig. 4. The first CPU round in Fig. 4 is same as Fig. 3. The difference starts from the second CPU round. The fourth seed, which is in the first PSO iteration, delays to the second CPU round. Instead of letting the other CPUs be idle, the seeds from the second PSO iteration are assigned to the remaining CPUs immediately. In this case, all CPUs are used all the time, and there is no computation resources idle. But the problem arises by nature: How can we decide the new seeds for second PSO iteration when the results from the first iteration has not completed?

The procedure in current PSO iteration is: 1) each seed calculates the objective function using the current parameter value in each seed; 2) once all seeds finish calculating objective function, the best optimal value will be chose from the seeds, say seed A; 3) new seed will be generated based on the current parameter value in seed A, and hence new velocity. In RCDF-AFC, we generate new seeds based on available partial results from first iteration.

To elaborate the procedure, we define a notation for a seed: $S^{(l)}_{i;j;k}$, where i represents ith PSO iteration, j stands for in jth seed, k is kth CPU round and l means that this seed is generated from the lth seed in the previous PSO iteration. For the PSO seed in the first iteration, l = 0. Using same logic, define $R^{(l)}{}_{i;j;k}$ as the optimal value calculated from seed $S^{(l)}{}_{i;j;k}$.

In the 1st CPU round, assume $R_{(0)1;3;1}$ is the best value among $R_{(0)\ 1;1;1}$, $R_{(0)\ 1;2;1}$, and $R_{(0)\ 1;3;1}$. We use the $S_{(0)\ 1;3;1}$ to generate the temporal seed for the 2nd PSO iteration. We define these seeds as "temporal seeds" since the $R_{(0)\ 1;3;1}$ is not determined to be the optimal value of first iteration before the result of $S_{(0)\ 1;4;1}$ is obtained in the second CPU round. In this example, there are 2 new temporal seeds generated for the 2nd PSO iteration in the 2nd CPU round. Once the 2nd CPU round finished, seed $S_{(0)1;4;2}$, $S_{(3)\ 2;1;2}$, and $S_{(3)\ 2;2;2}$ all finished. Now comparing the value $R_{(0)\ 1;3;1}$ and $R_{(0)1;4;2}$. If the value of $R_{(0)1;3;1}$ is optimal, the values of $R_{(3)2;1;2}$ and $R_{(3)2;2;2}$ are already calculated in the 2nd CPU round. In this case, the first two PSO iterations can be finished in totally 3 CPU rounds, and 1 CPU round is saved compared with the case in Fig. 3.

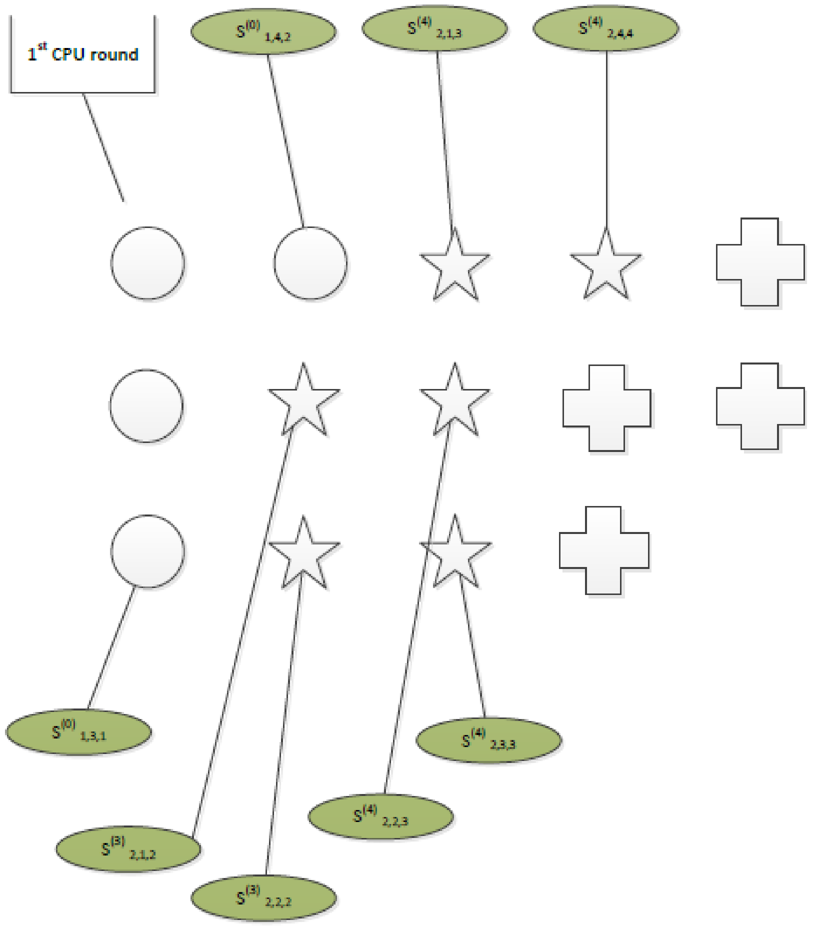


Figure 5. Y CPUs workflow on X seeds without idle status

The parallel degree of the system is increased and hence the performance is improved. If the value of $R_{(0)\ 1;4;2}$ is optimal, then four new seeds for the 2nd PSO iteration, $S_{(4)\ 2;1;3}$, $S_{(4)\ 2;2;3}$, $S_{(4)\ 2;3;3}$, and $S_{(4)2;4;4}$ are generated. The temporal seeds $S_{(3)\ 2;1;2}$ and $S_{(3)\ 2;2;2}$ are discard. Then the same process is applied to the 2nd PSO iteration and the 3rd CPU round in Fig. 5. We call this process as "Aggressive Forward Check" (AFC) since the seeds of (n+1)th iteration are aggressively computed before the completion of (n+1)th iteration. If the temporal seeds are demonstrated useful in the (n+1)th iteration, we call that AFC wins in the (n+1)th iteration; otherwise, we call that AFC misses since the temporary seeds are discarded.

Now we validate the performance gain of RCDF-AFC theoretically using probability in the decision tree. We use mod(A;B) to represent an operator which divides number A by B and returns only the remainder. For example, mod(4; 3) = 1.

We define Prb_i as the probability AFC wins in the ith PSO iteration. Note that the optimal value is uniformly generated among all the seed in one PSO iteration. Prb_2 can be expressed as:

$$Prb_2 = [X - \mathrm{mod}(X;Y)]/X, \tag{3}$$

Here we define $Prb_1 = 1$, or $TS_1 = Y$. In the example shown in Fig. 3, $Prb2 = [4 – \mathrm{mod}(4;3)]/4 = 3/4$. In another word, the probability of AFC wins in the 2nd iteration is as high as 75%, and the situation as in Fig. 5 only occurs at a probability of 25%. From perspective of statistics, AFC wins with a higher probability compared with AFC misses in this iteration.

We also define TS_i as the number of temporal seeds generated for (i+1)th PSO iteration based on the partial results from the ith PSO iteration. TS_1 can be expressed as:

$$TS_1 = Y - \mathrm{mod}(X;Y). \tag{4}$$

In the example shown in Fig. 3, $TS_1 = Y - \mathrm{mod}(X;Y) = 2$.

The dependency of Prb_i on TS_{i-1}, and the dependency of TS_i on TS_{i-1} can be generalized as:

$$Prb_i = [X - \mathrm{mod}(X – Ts_{i-1};Y)]/X, \tag{5}$$

$$TS_i = Y - \mathrm{mod}(X – Ts_{i-1};Y). \tag{6}$$

In Fig. 4, $Prb_3 = 0{:}5$ and $TS_3 = 1$. Generally, if $Prb_i = 1$, or $TS_i = Y$, it indicates that the system situation is same as 1st PSO iteration, i.e, $Prb_i = Prb_1$, and $TS_i = TS_1$. Note in Fig. 6, the whole process starts repeating from the 4th PSO iteration, which means $Prb_4 = 1$, and $TS_4 = 3$.

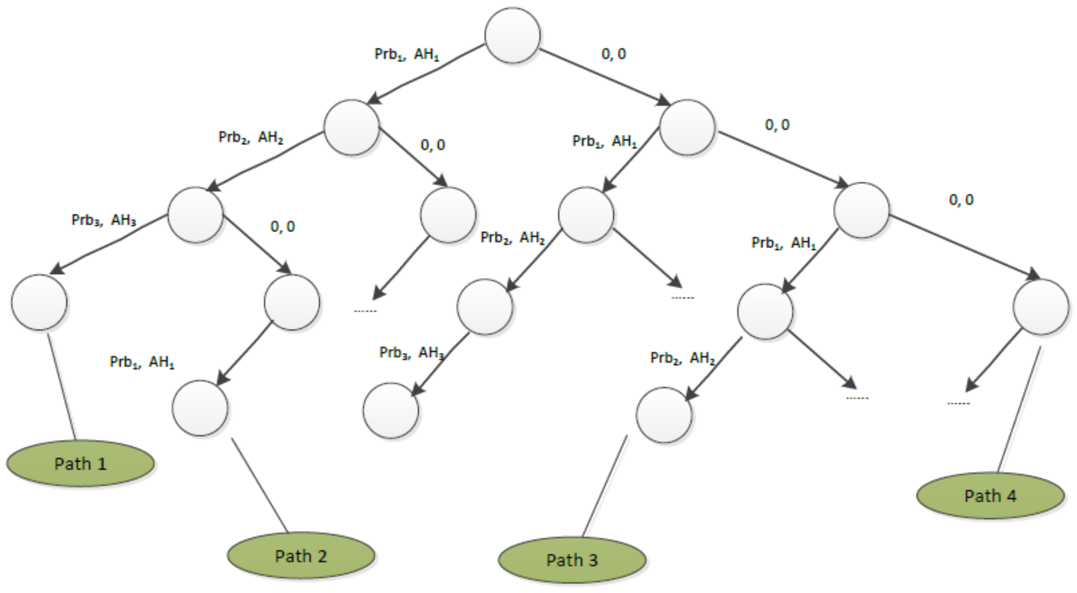


Figure 6. Probability analysis on AFC processing in PSO.

Fig. 6 shows that all of the combination of AFC process success or misses among consecutive PSO iteration. Path1 represents 3 consecutive successful AFC process. Path2 represents 2 consecutive successful AFC process, followed by 2 consecutive failed AFC process. Path3 represents 2 consecutive failed AFC process, followed by 2 consecutive successful AFC process. Path4 represents for 4 consecutive failed AFC process.

5. Simulation

MPC has been studied at least since the 1970s. At that time various works showed an incipient interest in MPC in the process industry [8][9]. 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 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 [10].

The general structure of MPC is shown in Figure 7. The mathematical model is formulated based on the actual system. Optimization problems are derived from the mathematical model, with explicit cost function and constraints. The result from the optimization problem is the input to the actual system again to obtain the next state and output. All the MPC algorithms possess common elements and different options can be chosen for each element giving rise to different algorithms.

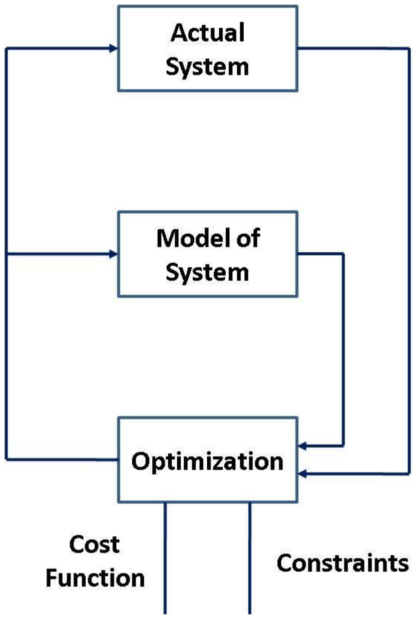


Figure 7. General Structure of Model Predictive Control.

We conducted the simulation in Matlab. PSO is used for optimization, as shown in Figure 7. We use a general unconstrained objective function to compare the performance of different optimization method.

$$obj : e_{x_1} \_ (4 \_ x_{21} + 2 \_ x_{22} + 4 \_ x_1 \_ x_2 + 2 \_ x_2 + 1). \qquad (7)$$

Table 1. Simulation results

	With AFC		Without AFC	
	mean	variance	mean	variance
Time	3.823585e-002	1.076474e-002	5.928038e-002	1.528491e-002
Value	1.718621e-013	2.524355e-029	3.660897e-015	0

From Table 1, we can conclude the accuracy between AFC and Quasi-Newton line search are almost the same. But the calculation time can be saved up to 35%.

6. Conclusion and future work

We have demonstrated a set of tools for analyzing a nonlinear MPC algorithm using PSO optimization, identifying bottlenecks, and suggesting ways to reduce the time needed for completing computations. The main point is that given an algorithm that needs to run in real time (not necessarily an MPC algorithm) to solve a class of problems, we need to explore the practical tools to improve the performance.

This paper especially explored hardware-software codesign using extended RCDF models. Although parallel systems are being developed exclusively, how to fully exploit all the computation capability is still an issue not only in high performance computing (HPC), but also in digital signal processing (DSP), embedded systems, and implementation of complicated control algorithms. Simulation results indicated that our work points to a good direction for a practical and efficient implementation.

We have not identified or exploited all the opportunities for parallelism in the algorithms we have analyzed so far. For example, the actor G can be expanded into a sub dataflow group, and we can explore more task level and data level parallelism inside the dataflow group.

It would be desirable to provide both an optimized algorithm for a large class of optimization problems and an efficient, reliable, and fast implementation of that algorithm. The result would be a kind of plug and play MPC. To do this one would need a good set of test candidates for MPC control and a reasonable collection of candidate algorithms. Our tools could then be applied to analyze the algorithms, find ways to parallelize them, and develop special purpose hardware to implement them efficiently and reliably. The resulting MPC controller, both linear and nonlinear, could greatly extend the applications of MPC in the real world.

7. References

[1] R. Gu, S. S. Bhattacharyya, and W. S. Levine, “Methods for efficient implementation of model predictive control on multiprocessor systems,” in Proceedings of the IEEE International Conference onControl Applications, September 2010.

[2] Z. Shi, C. Beard, and K. Mitchell, “Competition, cooperation, and optimization in multi-hop csma networks,” in Proceedings of the Eighth ACM International Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks (PE-WASUN 2011,October 2011.

[3] 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.

[4] Y. W. J. Mattingley and S. Boyd, “Receding horizon control: Automatic generation of high-speed solvers,” IEEE Control Systems Magazine, vol. 31, no. 3, June 2011.

[5] F. Manenti, G. Buzzi-Ferraris, I. Dones, and H. A. Preisig, “Generalized class for nonlinear model predictive control based on bzzmath library,” in Chemical Engineering Transactions 17, 2009, pp. 1209–1214.

[6] M. R. Fahey, Highly Parallel Linpack on the IBM p690*. Oak Ridge National Laboratory Technical Report, 2002.

[7] J. Kennedy and R.Eberhart, “Particle swarm optimization,” in Proceedings of Neural Networks, November 1995, pp. 1942–1948.

[8] 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.

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

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