Parallel Optimization Algorithms and Their Implementation in VLSI Design

G. Lee and J. J. Feeley
Department of Electrical Engineering
University of Idaho
Moscow, ID 83843

Abstract- **Two new parallel optimization algorithms based on the simplex method are described. They may be executed by a SIMD parallel processor architecture and be implemented in VLSI design. Several VLSI design implementations are introduced. An application example is reported to demonstrate that the algorithms are effective.**

1 Introduction

Optimal system control is an important part of modern control theory. The kernel problem is optimizing the behavior of systems, as in minimizing the energy or cost required to accurately reach some required terminal state. The search for the control which attains the desired objective while minimizing (or maximizing) a defined system criterion constitutes the fundamental problem of optimal control [1][2][3].

To date, practical applications of optimal control theory are still quite few in number. For a class of systems with fast response, the implementation of a real-time on-line optimal controller has been difficult. The time-consuming computation required for optimal control solutions has been a major obstacle. Modern supercomputers with parallel processing architectures and very fast computation speed are not a practical solution because of their weight, size and cost. Fast computation, small size and low cost are basic requirements for the controller. In this paper, the technique of an algorithmically specialized computer is suggested to achieve an optimal controller which can realize both real-time computation and on-line control for a rapidly responding system. Effective algorithms, parallel architecture, and VLSI implementation are involved in the design of the controller.

Efficient optimization algorithms are very necessary for solving the two-point boundary-value (TPBV) problems which arise in optimal control. Chazan and Miranker in 1970 [4] originally proposed a nongradient-based parallel search algorithm for unconstrained minimization which is suitable for execution using an array of parallel processors. The algorithm involves the parallel execution of n linear searches along the same direction, starting from n points, when the dimension of the vector of unknowns is n. Travassos and Kaufman [5] have applied the algorithm to the solutions of optimal control systems. Housos and Wing in 1984 [6] reported a parallel pseudo-conjugate direction algorithm that performs a set of n linear searches in parallel along different search directions. Those parallel optimization algorithms proceed by univariate optimization so that they are MIMD-type algorithms [7]. Although they may be used to solve the optimal control problem, it is not easy to shift them to VLSI design for a small size and low cost controller. Two new parallel,

nongradient-based algorithms for unconstrained optimization are presented in this paper. In contrast to existing parallel optimization algorithms, the new parallel algorithms are based on a simplex method and are SIMD-type algorithms [7]. The advantage of the new algorithms is that they do not need a linear search and may be easily shifted to VLSI implementation.

Three kinds of design schemes: digitally controlled analog, hybrid, and pure digital, are presented in this paper. Their VLSI implementation and their performances are discussed.

2 New parallel optimization algorithms

The following unconstrained minimization problem is considered:

$$\min f(X),\ X \in \Re^n,$$

where $f : \Re^n \rightarrow \Re$, and is usually non-quadratic and nonlinear.
We wish to find a point X^* numerically such that, if $\epsilon > 0$, then

$$f(X^*) < f(X), \text{ for all } X : \|X - X^*\| < \epsilon.$$

Two parallel simplex algorithms, PS1 and PS2, which are based on an improved simplex method [8][9] and use parallel function evaluations, are stated below.

Algorithm PS1: The algorithm PS1 predicts four candidate vertices simultaneously in one iteration. Therefore at least four parallel processors are required. Each iteration includes two phases: the first is for parallel evaluations and the second is for choosing a new vertex to generate a new simplex via function value comparison. The computation time for the function evaluations is always longer than the time for the function value comparison. The execution of parallel function evaluations effectively reduces computation time since it is a major part of the time for one iteration cycle. It is also important that the parallel function evaluations are of the SIMD type. This allows the algorithm to proceed in the SIMD parallel architecture. The number of parallel function evaluations required by PS1 is only about half the number required by the improved simplex algorithm of Nelder and Mead [8].

The algorithm PS1 is described below:

(0) Initial simplex:

(0a) Set the iteration number $k = 0$.

(0b) Starting point $v^0 = (x_1^0, x_2^0, \cdots, x_n^0)$ is given. An initial simplex $V^0 = [v_1^0, v_2^0, \cdots, v_{n+1}^0]$ is formed in parallel by: $v_1^0 = (1-\delta)v^0$

$$v_{i+1}^0 = \begin{cases} v^0 + \delta E_i x_i^0, & \text{if } x_i^0 \neq 0 \\ v^0 + \delta E_i, & \text{otherwise} \end{cases}$$

where $E_i = \{0 \ldots 0\ 1\ 0 \ldots 0\}, i = 1, 2, \ldots, n,$ and $\delta = 0.1$

(0c) Parallel evaluation of the function value at the vertices of V_0
$Y^0 = [f(v_1^0), f(v_2^0), \cdots, f(v_{n+1}^0)]$.

(1) Parallel sorting:

Set $k = k + 1$. Let $S^k = [X_1^k, X_2^k, \cdots, X_{n+1}^k]$ and $F^k = [f_1^k, f_2^k, \cdots, f_{n+1}^k]$ be ordered V^{k-1} and Y^{k-1}.

Find $d, d = \max(\|X_i^k - X_1^k\|), i = 2, 3, ..., n+1$, if d is small enough, then stop, otherwise continue as follows.

Denote

X_l by X_1^k, X_{sh} by X_n^k, X_h by X_{n+1}^k,

f_l by f_1^k, f_{sh} by f_n^k, f_h by f_{n+1}^k.

The centroid $\overline{X}$ is the mean of the vertices with $i \neq n+1$, i.e.,

$$\overline{X} = \frac{1}{n} \sum_{i=1}^{n} X_i^k$$

(2) Parallel computation:

$X_c = (1 - \beta)\overline{X} + \beta X_h$

$X_a = (1 + \beta)\overline{X} - \beta X_h$

$X_r = (1 + \alpha)\overline{X} - \alpha X_h$

$X_e = (1 + \gamma)\overline{X} - \gamma X_h$

Parallel function evaluations

$f_c = f(X_c)$, $f_a = f(X_a)$, $f_r = f(X_r)$ and $f_e = f(X_e)$

(3) Comparison and selection of new point for updating simplex:

(3a) If $f_e < f_r < f_l$, then $X_h = X_e$ and $f_h = f_e$.

(3b) If $f_{sh} > f_r > f_1$ or $f_e > f_r > f_1$, then $X_h = X_r$ and $f_h = f_r$.

(3c) If $f_h > f_r > f_{sh}$ and $f_a < f_{sh}$, then $X_h = X_a$ and $f_h = f_a$.

(3d) If $f_r > f_h$ and $f_c < f_{sh}$, then $X_h = X_c$ and $f_h = f_c$.

(3e) If $f_r > f_h$ and $f_c > f_{sh}$ or if $f_h > f_r > f_{sh}$ and $f_a > f_{sh}$, do shrinkage in parallel: $X_s = [X_1, X_{s_j}]$, where $X_{s_j} = (X_j + X_1)/2$, $j = 2, 3, \ldots, n+1$, evaluate and update $F^k = [f_1, f(X_{s_2}), f(X_{s_3}), \cdots, f(X_{s_{n+1}})]$ and $S^k = X_s$, then do

(4) Update the simplex:

let $V^k = S^k$, $Y^k = F^k$, then return to (1).

Algorithm PS2: The algorithm PS2 is developed from the algorithm PS1 by increasing the parallel processors to sixteen. Twenty processors in total are utilized to predict twenty candidate vertices simultaneously in one iteration. The algorithm PS2 is more effective than the algorithm PS1. One iteration of the algorithm PS2 is functionally equivalent to two iterations of the algorithm PS1. Thus the algorithm PS2 will do the same function in roughly half the time of the algorithm PS1. Algorithm PS2 is also of the SIMD-type.

The algorithm PS2 is described below:

(0) The same as Step (0) of Algorithm PS1;

(1) The same as Step (1) of Algorithm PS1;

(2) Parallel computation:

(2a) Compute the first level direction points (four in total) in parallel:

$X_c = (1-\beta)\overline{X} + \beta X_h$

$X_a = (1+\beta)\overline{X} - \beta X_h$

$X_r = (1+\alpha)\overline{X} - \alpha X_h$

$X_e = (1+\gamma)\overline{X} - \gamma X_h$

and find 4 conductive points in parallel

$\overline{X}_i = \frac{1}{n}(\sum_{j=1}^{n-1} X_j + X_i)$, $i =$'c', 'a', 'r', 'e',

(2b) Compute the second level direction points (sixteen in total) in parallel:

$X_{ic} = (1-\beta)\overline{X}_i + \beta X_{sh}$

$X_{ia} = (1+\beta)\overline{X}_i - \beta X_{sh}$

$X_{ir} = (1+\alpha)\overline{X}_i - \alpha X_{sh}$

$X_{ie} = (1+\gamma)\overline{X}_i - \gamma X_{sh}$

where $i =$'c', 'a', 'r', 'e',

(2c) Parallel function evaluations

$f_i = f(X_i)$

$f_{ic} = f(X_{ic})$, $f_{ia} = f(X_{ia})$, $f_{ir} = f(X_{ir})$ and $f_{ie} = f(X_{ie})$

where $i =$'c', 'a', 'r', 'e',

(3) Comparison and updating simplex:

Set $m = 0$.

(3a) If $f_e < f_r < f_l$, $j =$'e', or

if $f_{sh} > f_r > f_l$ or $f_e > f_r < f_p$, $j =$'r', or

if $f_h > f_r > f_{sh}$ or $f_a < f_{sh}$, $j =$'a', or

if $f_r > f_h$ and $f_c < f_{sh}$, $j =$'c', set $m = m+1$ and do (3b)

if $f_r > f_h$ and $f_c > f_{sh}$, or if $f_h > f_r > f_{sh}$ and $f_a > f_{sh}$, do shrinkage in parallel, $X_s = \{X_1, X_{s_j}\}$, where $X_{s_j} = (X_j + X_1)/2$, $j = 2, 3, \cdots, n+1$, evaluate $F_k = \{f_1, f(X_{s_2}), f(X_{s_3}), \cdots, f(X_{s_{n+1}})\}$ and let $s^k = X_s$, then do (4).

(3b) Replacing:
$X_h = X_{sh}$, $f_h = f_{sh}$, $X_{sh} = X_j$, $f_{sh} = f_j$, and $X_i = X_{ji}$, $f_i = f_{ji}$.
If $m = 1$ do (3a), otherwise do (4).

(4) The same as the step (4) of the algorithm PS1.

3 Example of application to real-time optimal control

The air-to-air missile-target intercept is a practical real-time optimal control application. A typical intercept mission from missile launch to intercept, may take only a few seconds. It is almost impossible to achieve true optimal control during such a short time interval with present technology. The efficiency of the algorithm PS2 for real- time application of optimal control is demonstrated in this section via simulation of a 3-dimensional air-to-air missile-target intercept problem. An optimal guidance law that minimizes missile energy expenditure with fixed final time tf and fixed final state (zero miss range) is derived in Ref [9] using nonlinear optimal control theory. This section focuses on solving the nonlinear TPBV problem (NTPBVP) which arises in the intercept problem by the "shooting method" using Algorithm PS2.

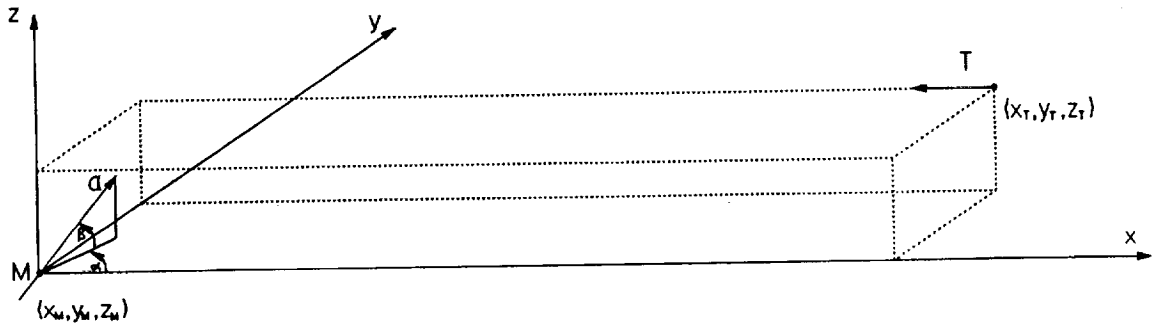

Figure 1: 3-dimensional intercept geometry

Figure 1 shows the 3-dimensional intercept scenario. The target **T** moves in a straight line at constant velocity v_T and the missile **M** moves at controlled acceleration $a(t)$ and its direction angles are $\alpha(t)$ and $\beta(t)$. An on-board optimal controller in the missile calculates and provides, $a(t)$, $\alpha(t)$ and $\beta(t)$ to the missile thruster.

The NTPBVP obtained is

$$
\begin{array}{lcl|lcl|lcl}
\dot{x}_1 &=& x_4 & x_1(0) &=& x_{10}, & x_1(t_f) &=& 0 \\
\dot{x}_2 &=& x_5 & x_2(0) &=& x_{20}, & x_2(t_f) &=& 0 \\
\dot{x}_3 &=& x_6 & x_3(0) &=& x_{30}, & x_3(t_f) &=& 0 \\
\dot{x}_4 &=& -a(x_{10}^2 + x_{11}^2)x_{10} & x_4(0) &=& x_{40} & & & \\
\dot{x}_5 &=& -a(x_{10}^2 + x_{11}^2)x_{11} & x_5(0) &=& x_{50} & & & \\
\dot{x}_6 &=& -ax_{12} & x_6(0) &=& x_{60} & & & \\
\dot{x}_7 &=& 0 & & & & & & \\
\dot{x}_8 &=& 0 & & & & & & \\
\dot{x}_9 &=& 0 & & & & & & \\
\dot{x}_{10} &=& x_7 & & & & x_{10}(t_f) &=& 0 \\
\dot{x}_{11} &=& x_8 & & & & x_{11}(t_f) &=& 0 \\
\dot{x}_{12} &=& x_9 & & & & x_{12}(t_f) &=& 0
\end{array}
\tag{1}
$$

where

$$a = (x_{10}^2 + x_{11}^2)^2 + x_{12}^2.$$

Notice that the initial conditions x_{10} to x_{60} are constant and the terminal values $x_1(t_f)$ to $x_3(t_f)$ and $x_{10}(t_f)$ to $x_{12}(t_f)$ are zeros. The first six equations of (1) are the dynamics of the system. The second six equations are the co-state equations. The shooting method starts with estimating a set of initial values $(x_7(0)x_8(0)x_9(0))^T$, then integrates (1) forward, with given and estimated initial values $x_1(0)$ to $x_{12}(0)$. The resulting terminal values are usually different from the given ones. An error function E is defined by

$$E = \sqrt{x_1(t_f)^2 + x_2(t_f)^2 + x_3(t_f)^3 + x_{10}(t_f)^2 + x_{11}(t_f)^2 + x_{12}(t_f)^2} \tag{2}$$

The shooting method attempts to minimize the error function E:

$$\min E \to 0 \tag{3}$$

This can be done by means of the algorithm PS2 to update the estimated initial values until (3) is satisfied.

The initially given condition is [1]

$$
\begin{array}{lcrl}
x_{10} &=& 20000 & (ft) \\
x_{20} &=& 3000 & (ft) \\
x_{30} &=& 2500 & (ft) \\
x_{40} &=& -972 & (ft/sec) \\
x_{50} &=& -972 & (ft/sec) \\
x_{60} &=& 0 & (ft/sec)
\end{array}
$$

and the fixed final time is $t_f = 5(sec)$.

Assume the target velocity v_T is constant, the travel path of the target will be a straight line. The target path may be calculated correctly by

[1] Data taken from Ref. [10]

$$\begin{aligned} x_T(t) &= x_{0T} + v_T t \\ y_T(t) &= y_{0T} \\ z_T(t) &= z_{0T} \end{aligned}$$

where $[x_{0T}\ y_{0T}\ z_{0T}]^T$ is the target's initial position so that open-loop optimal control may be employed by the missile.

To come up with open-loop optimal control numerically, one must first solve the NTP-BVP (1). For a set of rough initial estimations

$$\begin{aligned} x_{70} &= 1 \\ x_{80} &= 1 \\ x_{90} &= 1 \end{aligned}$$

using the algorithm PS2, the resulting solutions are in Table 1:

TI (sec)	OIV	PFE	CMR (ft)	RMR (ft)
[0 5]	$x_7(0)$=0.65993 $x_8(0)$=–0.08107 $x_9(0)$=0.92175	114	0	1.287e–9

Table 1: The numerical results of the intercept scenario

TI–Time interval,
OIV–Optimal initial values,
PFE–Parallel function evaluations,
CMR–Constraint of miss range,
RMR–Real miss range.

As a rough estimation of computation time, if the PFE < 120 in five seconds as shown in Table 1, then the real-time optimal control can be implemented for this air-to-air missile-target intercept problem. This is very possible with modern VLSI techniques. In the next section several VLSI design possibilities are introduced.

4 VLSI implementations

This section presents design possibilities for potential real-time, on-line, optimal controllers. These optimal controllers will be algorithmically specialized parallel computers

consisting of a few VLSI chips. Small special-purpose optimal controllers should be useful for certain optimal control systems, such as aircraft control, missile guidance, etc.

To conserve space, only the algorithm PS1 is considered for VLSI implementation in this section. The design procedure can be used for algorithm PS2, but the resulting circuit will be more complex.

4.1 Schematic design

A schematic diagram of the implementation of the algorithm PS1 is shown in Figure 2. The dashed box performs the main function of the algorithm PS1. In order to be useful for various control systems, a parallel function evaluator (PFE) is separated from the dashed box. The PFE is an array of four [2] parallel processors. The complete system includes two separate parts: the main algorithm part and the PFE. The main part is the algorithm itself in which the design is fixed. The PFE is more flexible and is different from system to system.

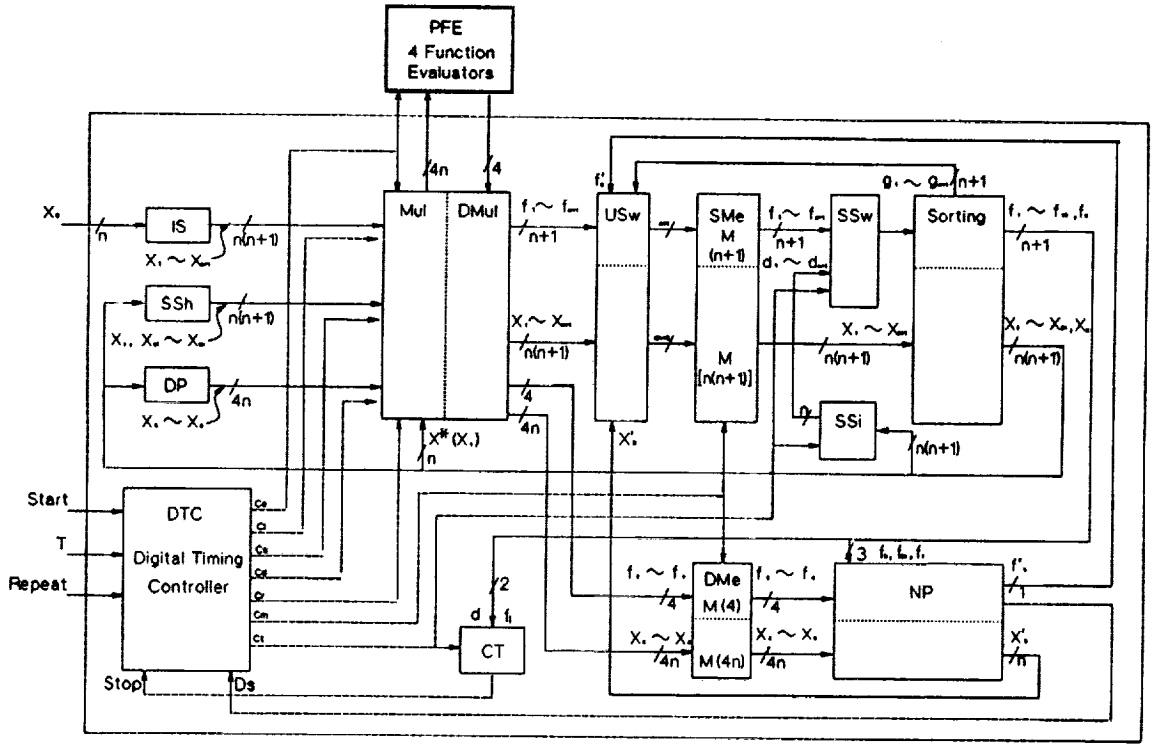

Figure 2: Block diagram of the algorithm PS1

The operation of the system outlined in Figure 2 may be described as follows.

The IS, connected to the input X_0, is for the generation of an initial simplex $[X_{01} \cdots X_{0(n+1)}]$. Via the multiplexer (Mul), the function values on the initial simplex may be evaluated by the external PFE. The outputs of the PFE, $[f(X_1), \cdots, f(X_{n+1})]$, via the

[2] Twenty for the algorithm PS2.

demultiplexer (DMul) and a set of updating switches (USw), are saved in the simplex memories (SMe). The Mul and the DMul also pass the initial simplex vertexes, denoted as $[X_1 \cdots X_{n+1}]$, to the SMe. Then a basic simplex with its function values is stored for further operations.

According to the algorithm PS1, the stored simplex must be updated. To do this, the simplex in the SMe must be first sorted by a sorting circuit (Sorting). A sorted simplex, $[X_l, \cdots, X_{sh}, X_h]$ with function values $[f_l, \cdots, f_{sh}, f_h]$, is available at the output of the Sorting. From it four direction points, X_c, X_a, X_r and X_e, can be found in the direction points module (DP). Similar to the initial simplex, they and their function values, f_c, f_a, f_r and f_e, evaluated by the PFE are stored in the direction memories (DMe) via the Mul and the DMul. A new point module (NP) compares [f_c, f_a, f_r and f_e] with [f_l, f_{sh} and f_h] and then selects a proper one of the direction points, denoted by X'_h, with its function value f'_h. Via the USw the X'_h replaces the vertex X_h to update the basic simplex. The positions of X_h and f_h are indexed by one of the signals g_1 to g_{n+1} generated by the Sorting.

In case no new point can be selected, the NP will send out a digital signal Ds. Through it the DTC generates another control signal Cs to the Mul and the DMul, then a shrunken simplex from the simplex shrinkage (SSh), $[X_1, X_{s1}, \cdots, X_{sn}]$, with its function values is passed to the SMe, so that the basic simplex is updated.

The simplex size module (SSi) and the convergence testing module (CT) monitor the size of the sorted simplex and its minimal function value. Together with the size switches (SSw) and the Sorting, when one of them satisfies a given criterion, the CT will send a "stop" signal to finish the iterations.

The digital timing controller (DTC) is necessary to control the timing of the whole system. The functions of the DTC may be stated by defining its inputs and outputs as follows:

Inputs:

Start:	actuates the DTC and starts the computation,
T:	a parameter given for setting up the width of the Ce's active interval,
Repeat:	after the computation, reactuates the DCT and repeats the solutions if necessary,
Stop:	stops the iteration when the solutions are available,
Ds:	active when shrinkage simplex is needed, sets up the Cs,

Outputs:

Ce:	actuates the PFE, the Mul and the DMul, its active length is given by the input T,
Ci:	passes the initial simplex and its function values to the SMe via the Mul and the DMul,
Cs:	passes the shrinkage simplex and its function values to the SMe via the Mul and the DMul, it is controlled by the input Ds,
Cd:	passes the direction points and their function values to the DMe via the Mul and the DMul,
Cr:	repeats the computation, it is controlled by the input "Repeat",
Cm:	actuates the SMe and the DMe,
Ct:	tests the simplex size, active at each iteration.

The design of the DTC is a normal digital logic design and is not included here.

To meet various application requirements, three kinds of design schemes (1) digitally controlled analog, (2) hybrid, and (3) all-digital, are suggested here. The design of digitally controlled analog is due to analog computation on both the PFE and the main algorithm part. The hybrid design uses digital computation for the main algorithm. Finally the digital design is a pure digital scheme. Due to their different characteristics, they are employed in different circumstances as listed below.

For the digitally controlled analog controller:
1. Accuracy limited but faster computation.
2. Limited memory period.
3. More efficient for low frequency systems with shorter operation time.

For the hybrid controller:
1. Same as 1 above.
2. Unlimited memory period.
3. More efficient for low frequency systems with longer operation time.

For the digital controller:
1. Accuracy unlimited but slow computation.
2. Unlimited memory period.
3. No strong relation to frequency and time of system operation.

4.2 Digitally controlled analog scheme

In general, analog computation is faster than digital computation. This suggests the PFE and the main algorithm part (not including the DTC) may be implemented by analog techniques. However analog long-time memory is not easily implemented on a VLSI chip. Memory time is strongly depended on the problem's complexity. If the requirement for memory time is too long and the size requirement is critical, the hybrid computation scheme should be considered as below.

4.3 Hybrid scheme

The hybrid scheme includes digital computation and memories in the main algorithm part. But the PFE still uses analog techniques. In practice, the PFE is a parallel electronic differential analyzer (EDA) which consists of some integrators. Integration computations is more convenient with analog circuiting than with digital. Keeping the PFE in analog will reduce computation time. A hybrid scheme is suggested in Figure 3. The system has three sections: the PFE, the digital algorithm processor and the linkage system, in which some analog/digital (A/D) and digital/analog (D/A) converters are essential.

Each numerical value in the digitally controlled analog scheme mentioned above, such as each function value, each element value of a simplex vertex and each constant value, will be described in m-bit form and be stored in a m-bit register. In order to achieve parallel computation, the input and output to these registers are parallel m-bit data buses. Furthermore, all of the digital devices in this system are parallel.

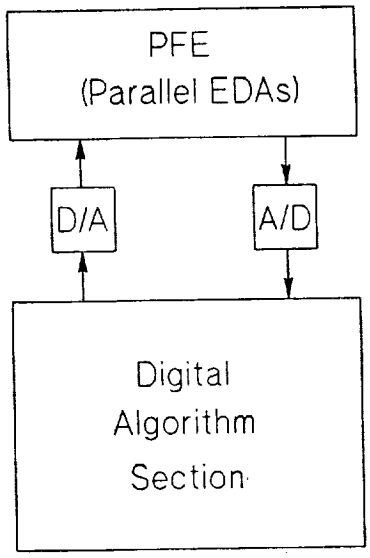

Figure 3: Hybrid scheme

4.4 Digital scheme

Based on the hybrid scheme, a pure digital optimal controller may be obtained by designing a digital PFE. A key point is to design, for the PFE, a digital integrator, which is very different from an analog one. The design of the digital PFE is related to both the solution methods and the particular problem, and may be separated into two parts. The first one is an algorithmically specialized unit of a solving algorithm, such as the Runge-Kutta algorithm, and another is a computing unit of a given differential equation.

5 Conclusion

Two new parallel optimization algorithms, PS1 and PS2, based on the simplex method are described. Four processors are required for PS1 and twenty for PS2. They may be executed by a SIMD parallel processor architecture and may be easily shifted to VLSI design.

The numerical result of a 3-dimensional air-to-air missile-target intercept problem has been reported to demonstrate that the algorithms are effective and the real-time optimal controllers are feasible for a class of optimal control systems with fast response.

As a design example, the algorithm PS1 has been shifted to a VLSI implementations. Three types of controller design schemes have been presented: (1) digitally controlled analog, (2) hybrid, and (3) pure digital controller. They can be employed satisfactorily for different application requirements.

In general, the optimal controllers converge rather rapidly, once the estimation of an initial value is found such that the evaluation of the error function E being minimized results in a number in the neighborhood of zero. However, if the problem to be solved is very sensitive to small perturbations in the initial co-state vector, convergence to an optimal

solution may be slow, or even fail. This case was not considered in this research. To overcome this problem a method [5] suggested by R. Travassos and H. Kaufman may be added in the design of the optimal controllers. This approach is currently under consideration.

References

[1] F. L. Lewis, Optimal Control, New York: Wiley, 1986.

[2] Andrew P. Sage and Chelsea C. White, III, Optimum Systems Control, Second edition, Prentice-hall., Englewood Cliffs, New Jersey, 1977.

[3] A. E. Bryson, Jr. and Yu-Chi Ho, Applied Optimal Control, Blaisdell Publishing Company, 1969.

[4] D. Chazan and W. L. Miranker, A Nongradient and Parallel Algorithm for Unconstrained Minimization, *SIAM J. Control*, Vol.8, No.2, pp.207-217, May 1970.

[5] R. Travassos and H. Kaufman, Parallel Algorithms for Solving Nonlinear Two-Point Boundary-Value Problems Which Arise in Optimal Control, *Journal of Optimization Theory and Applications*, Vol.30, No.1, pp.53-71, Jan. 1980.

[6] E. C. Housos and O. Wing, Pseudo-Conjugate Directions for the Solution of the Nonlinear Unconstrained Optimization Problem on a Parallel Computer, *Journal of Optimization*, Vol.42, No.2, pp.169-180, Feb. 1984.

[7] S. Lakshmivarahan and Sudarshan K. Dhall, Analysis and Design of Parallel Algorithms: Arithmetic and Matrix problems, McGraw-Hill, Inc., 1990.

[8] J. A. Nelder and R. Mead, A Simplex Method for Function Minimization, *Computer Journal*, Vol. 7, pp.308-313, 1965.

[9] G. Lee, Parallel Computation for Optimal Control Systems, University of Idaho, Ph.D. Dissertation, 1991.

[10] G. M. Anderson, Comparison of Optimal Control and Differential Game Intercept Missile Guidance Laws, *J. Guidance and Control*, Vol.4, No.2, pp.109- 115, March-April 1981.