1 Introduction
The design of complex engineering systems often requires the involvement of a variety of experts. These experts may originate from different disciplines or even the same discipline. As experts, they typically understand a specific part of the aggregate design problem. In order to provide a structure for these experts to communicate and coordinate their actions, a variety of approaches have been created, which broadly fall under the field of multidisciplinary design optimization (MDO).

Regardless of the approach chosen, the amount of time spent in the design process is of critical importance. It is very often the case that a design process does not end when the optimal solution is found but when time runs out. The importance of managing time in a design process has been recently underscored by the challenges faced by Boeing during the design of the 787 Dreamliner [1,2], and it is estimated that delays cost Boeing at least 100 orders [3]. The product development challenges faced by Boeing are complex, but their severity emphasizes the significant cost incurred by delays in product delivery.

Even in less extreme examples, time remains a critical resource in a design process, whether it means minimizing time to market or meeting an important development deadline [4]. Understanding the equilibrium stability of design problems can reduce time wasted by designers who may never agree to a compromise solution. In this paper, stability describes the steady state behavior of a distributed design problem. An equilibrium can be described as convergent, divergent, or a saddle point. The dynamic behavior of a design system is described by its transient response, which encompasses the system’s convergence time and the shape of the convergence curve.

One of the benefits of this work is that it expands the applicability of systems theory to distributed design problems. It explores and addresses the theory’s previous limitations as they relate to process architecture and provides an accurate assessment of system stability. This assessment can be used as a coarse filter to differentiate between convergent and divergent architectures. Further, this work provides direct insight into the mechanics governing convergence in distributed design.

The Impact of Process Architecture on Equilibrium Stability in Distributed Design

In distributed design processes, individual design subsystems have local control over design variables and seek to satisfy their own individual objectives, which may also be influenced by some system level objectives. The resulting network of coupled subsystems will either converge to a stable equilibrium or diverge in an unstable manner. In this paper, we study the dependence of system stability on the solution process architecture. The solution process architecture describes how the design subsystems are ordered and can be either sequential, parallel, or a hybrid that incorporates both parallel and sequential elements. In this paper, we demonstrate that the stability of a distributed design system does indeed depend on the solution process architecture chosen, and we create a general process architecture model based on linear systems theory. The model allows the stability of equilibrium solutions to be analyzed for distributed design systems by converting any process architecture into an equivalent parallel representation. Moreover, we show that this approach can accurately predict when the equilibrium is unstable and the system divergent when previous models suggest that the system is convergent.

[DOI: 10.1115/1.4004463]

Keywords: distributed design, equilibrium, process architecture
on object decomposition, aspect decomposition, sequential decomposition, or model based decomposition [11]. A survey of the relative merits of these decomposition approaches is performed by Sobieszczanski-Sobieski and Haftka [5].

The second step is the development of the design communication and coordination framework. MDO frameworks specify the mechanics of how the design problem is solved including the subsystem objective functions, communication protocols, design variable control, and other coordination procedures required for the decomposed subsystem to effectively iterate to a solution. There are a wide range of MDO frameworks, which provide these requirements while making certain guarantees about system convergence and the optimality of the final converged solution. These approaches include analytic target cascading [12], concurrent subspace optimization [13,14], bilevel integrated system synthesis [15], and collaborative optimization [16].

Each of these frameworks has its own advantages. For example, analytic target cascading guarantees that the decentralized system converges and that the converged value is globally optimal [12]. It also provides for design process traceability and facilitates the integration of marketing, business, and design systems [17]. In spite of the advantages offered by existing MDO frameworks, there are many cases when a formal framework is not used.

There are several reasons why a MDO framework may not be applied to a complex design problem. Applying a framework requires a significant level of coordination between subsystems and a high level of management expertise. Further, the engineering and design personnel involved must all agree to some extent to the proposed decomposition and framework. Achieving agreement is challenging when the individual designer interests do not align with those of the system level optimizer and when the strict protocols required by most MDO methods cannot be enforced. There are also some cases that do not naturally lend themselves to formal decomposition or where the parties involved cannot agree on an appropriate framework. When no formal framework is chosen or when the framework does not specifically prescribe subsystem interactions, the design problem framework may simply become a distributed design problem. The assumptions and mechanics governing distributed design problems are discussed in Sec. 3.

3 Distributed Design

Distributed design processes can be understood and analyzed through the consideration of two primary characteristics: (1) equilibrium stability and (2) transient response. This work focuses on the relationship between equilibrium stability and process architecture. Before considering this relationship, the underlying problem structure of distributed design processes is presented in Sec. 3.1 and their equilibrium stability is discussed in Sec. 3.2. Process architecture is discussed in Sec. 3.3 and related to stability in Sec. 3.4.

3.1 Problem Structure. Distributed design problems are a specific type of MDO problem and are typically nonhierarchical with a set of different designers, or subsystems, each seeking to optimize their own individual objectives. An additional property of the distributed design problems examined in this paper is that they are noncooperative; the subsystems are only compelled to share information that is absolutely necessary for the completion of the solution process. Although some MDO frameworks may share other types of information like behavioral or state variables, organizational barriers, computational limitations, or confidentiality concerns between noncooperative designers preclude them from willingly disclosing information other than their solution, as represented by the design variables. A set of four assumptions that capture this level of design information sharing is outlined in Ref. [18] and forms the basis for information sharing in this paper. These assumptions are as follows:

1. Designers have knowledge of only their own local objectives;
2. Designers act unilaterally to minimize their objective function;
3. Designers have complete control over specific local design variables; and
4. Designers communicate by sharing the current value of their local design variables.

The applicability of these assumptions to decentralized design problems is discussed in various contexts in Refs. [19-22] using examples that include the design of passenger aircraft, automotive engines, semiconductor chips, and steam turbines. Distributed design problems can also emerge as iterative subproblems in a larger MDO process. For example, in Refs. [23,24], the ordering of decomposed design systems was examined and iterative loops emerged due to subsystem coupling of concurrently executed tasks.

When subsystems iterate under the assumptions of noncooperation, the equilibrium stability and transient response are two fundamental concerns. The system transient response is an important area of research and has been investigated in Refs. [25,26]. In this paper, we focus on the equilibrium stability of distributed design systems to provide a broad perspective of their convergence properties.

3.2 Equilibrium Stability. Understanding the convergence behavior of design systems is important for creating high quality engineered products and systems in a timely manner. It has been suggested in Refs. [27,28] that there is a link between the performance of design teams and their convergence to a common problem representation, goals, or solution. The convergence of design teams is further examined in Ref. [29] where the relationship between initial information and team convergence is considered. These studies provide insight into the convergence of design teams, and they demonstrate that not all design teams converge. In Ref. [29], some teams diverged and could not reach a mutually acceptable solution.

Identifying these divergent design scenarios is an important area of investigation and has been a topic of research for some time in distributed design processes. The initial work on stability was performed by Vincent [19] for two-designer, two design variable problems, which models designers as players in an iterative game using game theory. Game theoretic models were further investigated by Lewis and Mistree [20] and were recently used by Takai [30] to investigate projects where designers must perform both individual and team functions. In the work by Vincent, each player alternates minimizing their objective function and communicates the associated design variables to the other player. Each step in this alternating process is a play in a sequential game. After repeated play of the sequential game, the players either converge and stop playing or diverge and continue playing indefinitely. When the players converge, they converge to a specific point called the Nash, or noncooperative, equilibrium [31]. The Nash solution can be described mathematically for a two player game with a set of solutions described by the vector pair \( (x_1, x_2) \). This solution pair is a Nash solution, \( (x_{1N}, x_{2N}) \), if they fulfill the requirements outlined in Eq. (1).

\[
F_1(x_{1N}, x_{2N}) = \min_{x_1} F_1(x_1, x_{2N})
\]

\[
F_2(x_{1N}, x_{2N}) = \min_{x_2} F_2(x_{1N}, x_2)
\]

In Eq. (1), \( F_1 \) and \( F_2 \) are the objective functions for player 1 and player 2 who control design variables \( x_1 \) and \( x_2 \), respectively. A solution pair \( (x_1, x_2) \) that meets the criteria in Eq. (1) is a Nash solution because the pair is a minimum for both \( F_1 \) and \( F_2 \). Although, in game theory, the participants in a game are called players, in engineering design, they are typically embodied as designers or subsystems. The relationship demonstrated in Eq. (1) can be understood qualitatively as the point at which no subsystem can unilaterally improve its objective function [32]. In Eq. (1), the Nash solutions are identified through an optimization
formulation, but they can also be expressed as the intersection of two sets defined by Eq. (2).

\[ (x_{1N}, x_{2N}) \in X_{1N}(x_2) \times X_{2N}(x_2) \]

\[ X_{1N}(x_2) = \{ x_{1N}|F_1(x_{1N}, x_2) = \min_{x_1} F_1(x_1, x_2) \} \]

\[ X_{2N}(x_2) = \{ x_{2N}|F_2(x_1, x_{2N}) = \min_{x_2} F_2(x_1, x_2) \} \]

The sets \( X_{1N} \) and \( X_{2N} \) are the rational reaction sets (RRSs), also called the best response set in Refs. [33,34], belonging to subsystem 1 and subsystem 2, respectively. These sets embody all the possible reactions that a subsystem may have toward a decision made by another subsystem. While determining a subsystem’s RRSs is not a trivial task, methods have been developed to approximate them for large systems [35]. The intersection of the RRSs for all subsystems is by definition the Nash equilibrium.

The concepts of RRSs and Nash equilibrium are demonstrated graphically in Fig. 1 for the problem summarized in Eq. (3). In Eq. (3), subsystem 1’s objective function is described by \( F_1 \) and it controls design variable \( x \). Subsystem 2’s objective function is described by \( F_2 \) and it controls design variable \( y \). For unconstrained optimization problems such as these, the RRSs can be determined by setting the gradient of the objective function with respect to the local design variables equal to zero. As seen in Fig. 1, the repeated plays of the game converge to the Nash equilibrium at \((x, y) = (-0.128, -0.927)\), defined by the intersection of the two subsystem’s RRSs.

\[ F_1 = 18.37x^2 - 5.19xy + 2.19y^2 - 0.09x - 11.59y \]

\[ \frac{\partial F_1}{\partial x} = 36.74x - 5.19y - 0.09 = 0 \]

\[ F_2 = 3.46y^2 + 12.44xy - 18.21x + 8.03y \]

\[ \frac{\partial F_2}{\partial y} = 6.92y + 12.44x + 8.03 = 0 \]

General conditions to evaluate stability in two-subsystem unconstrained quadratic systems were developed by Chanron and Lewis [36]. Conditions were also developed for systems with higher order models and nonlinear rational reaction sets in Refs. [37,38]. Although the work by Chanron and Lewis was assumed to be equally applicable to games with sequential and simultaneous play, the developments presented in Sect. 3.3 demonstrate that it is only applicable to games with simultaneous play. Smith and Eppinger [39] also studied games with simultaneous play and independently demonstrated principles similar to those found by Chanron.

A recent extension of this convergence work was performed by Gurnani and Lewis who demonstrated that the introduction of “mistakes” into the design process could cause some systems identified to be divergent using the approach by Chanron to converge to a solution [40]. The emphasis of this paper is on the ordering of the solution process, which has not been considered in the previous work.

### 3.3 Solution Process Architecture

In the context of a solution process, architecture refers to the ordering or organization of how the design subproblems are solved. It does not refer to the architecture of the product itself, which is an independent and significant area of design research [41]. Instead, it refers to the structure of the solution process, which includes both sequential and simultaneous elements. In Fig. 2, the difference between a purely sequential architecture, a purely simultaneous architecture, and a hybrid approach utilizing both sequential and parallel elements is illustrated.

Each of the three process architectures in Fig. 2 represents a single iteration of a solution process. Subsystems repeat these iterations in order to solve for specific values of system design variables. Within each iteration, there can be multiple stages where a single subsystem in sequence or group of subsystems in parallel executes their solution. Groups of subsystems in parallel execute their solution assuming that the group’s design variables remain unchanged from the previous iteration. The difference between iteration, stage, and subsystem is shown for the hybrid architecture in Fig. 2. The number of stages in a process architecture depends on the number of subsystems and the process architecture chosen. For purely sequential process architectures, the number of stages is equal to the total number of subsystems. In contrast, purely simultaneous, or parallel, architectures always consist of a single stage. The number of stages for sequential and parallel process architectures provides an upper and lower bound, respectively, for the number of stages in hybrid process architectures. For example, the hybrid process architecture in Fig. 2 has two stages.

To evaluate the impact of process architecture on the transient response of distributed design processes, we analyze sequential and parallel process architectures for the system described by Eq. (4) and discussed by Vincent [19].

\[ F_1 = x^2 + xy - 3x \]

\[ \frac{\partial F_1}{\partial x} = 2x + y - 3 = 0 \]

\[ F_2 = 0.5y^2 - xy \]

\[ \frac{\partial F_2}{\partial y} = y - x = 0 \]

In the two subsystem design process outlined by Eq. (4), subsystem 1 controls design variable \( x \) and subsystem 2 controls design variable \( y \). Their objective functions are described by \( F_1 \) and \( F_2 \), respectively, and the design variables are initialized to \((x, y) = (1.2)\). The convergence criterion for the system is a minimum design variable change of 2% as measured from the initial
impacts the system stability by changing the aggregate system eigenvalues. These eigenvalues have been previously believed to be independent of the process architecture. However, we propose the following hypothesis: The chosen process architecture impacts the system stability by changing the aggregate system eigenvalues.

To study this hypothesis, an experiment was performed using a set of randomly generated distributed design systems. In order to facilitate the representation of larger design systems, matrix notation is introduced for these systems. The notation used in this paper is the same notation used by Chanron and Lewis [36]. The system shown in Eq. (4) is first generalized to represent quadratic systems of n designers [36], as shown in

\[ F_n = X^T A X + Y^T B Y + X^T C Y + D Y + E Y + F \]  

In this representation of the nth subsystem’s quadratic objective function, \( F_n \), \( X \) is a vector of length \( i \), which contains the \( i \) local design variables, while \( Y \) is a vector of length \( j \), which contains the \( j \) nonlocal design variables. The coefficients associated with the second order elements of \( F_n \) for the local design variables are contained in the diagonal \( i \times i \) matrix \( A \) while the coefficients associated with the nonlocal design variables are contained in the \( j \times j \) matrix \( B \). In this representation, the \( A \) matrix is formulated as a diagonal to decouple the subsystem’s local design variables from one another. This guarantees each design variable value can be determined independently and a specific RRS can be formulated for each design variable. When these variables are coupled, the design system can still be represented using the form in Eq. (5) using a change in variable. The representation in Eq. (5) is examined in more depth in Ref. [36].

Although the local design variables must be decoupled, it is acceptable for the local and nonlocal design to be coupled with one another through the coefficients in the \( i \times j \) matrix \( C \). The terms in the \( C \) matrix describe the coupling between different subsystems. The remaining two vectors in Eq. (5) capture the linear elements of the system for the local and nonlocal design variables and have length \( i \) and \( j \), respectively. The term \( D \) is simply a scalar and does not play a significant role when analyzing the system stability. The important elements in Eq. (5) emerge when the gradient of the subsystem is taken with respect to its local design variables. Setting the gradient equal to zero results in \( i \) independent equations that represent the designer’s RRS. After the RRSs are found for each subsystem, there are \( mn \) equations, where \( m \) is the total number of design variables controlled by all subsystems in the distributed design problem. The RRS is shown in vector form in

\[ \frac{\partial F_n}{\partial X} = 2A X + CY + D = 0 \]  

The RRS in Eq. (6) specifies how each of the system’s \( n \) subsystems will respond to changes in design variable values and suggests that the system’s overall transient response is related to the parameters \( A \), \( C \), and \( D \) for each subsystem. Using these parameters, Chanron developed the discrete state space based representation to model the subsystems collectively using the update relationship in Eq. (7). The resulting stability criterion is shown in Eq. (8) using the variables defined in Eqs. (9a) and (9b)

\[ X_{s+1}^t = \Phi X_s^t + \Gamma \]  

\[ r_s(\Phi) < 1 \]  

\[ \Phi = -\frac{1}{2} \begin{bmatrix} A_1^{-1} & 0 & \cdots & 0 \\ 0 & A_2^{-1} & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & A_m^{-1} \end{bmatrix} \begin{bmatrix} C_{12}^T & \cdots & C_{1m}^T \\ \vdots & \ddots & \vdots \\ C_{m2}^T & \cdots & C_{mm}^T \end{bmatrix} \]  

\[ \Gamma = \frac{1}{2} \begin{bmatrix} A_1^{-1} D_1^T \\ \vdots \\ A_m^{-1} D_m^T \end{bmatrix} \]  

In Eq. (7), the subscript \( s \) denotes that \( X_s \) is a vector comprising all the system design variables and the superscript denotes the iteration number, which is consistent with linear system theory. Since Eq. (7) describes the interactions between subsystems, \( X_{s+1}^t \)
is length \( m \) containing all the design variables controlled by the subsystems. The design variable values at the \((k + 1)\)th iteration are a function of the previous design variable values at the \(k\)th iteration; they are expressed as \( X_k \) multiplied by a matrix \( \Phi \) plus a constant \( \Gamma \). The derivations for \( \Phi \) and \( \Gamma \) can be found in Ref. [36] and are summarized in Eqs. (9a) and (9b).

The matrix \( \Phi \) captures design variable interactions between quadratic elements found in the \( A \) and \( C \) matrices while the vector \( \Gamma \) captures interactions between quadratic and linear elements found in the \( A \) and \( D \) matrices, respectively. To populate \( \Phi \) and \( \Gamma \), the appropriate \( A \), \( C \), and \( D \) matrices are used and can be determined by examining which subsystem controls the design variable associated with the row being populated. The resulting dimensions for \( \Phi \) and \( \Gamma \) are \( m \times m \) and \( m \times 1 \), respectively. When examining system stability, only the \( \Phi \) matrix needs to be considered, and the criteria in Eq. (8) specify that, for stable systems, \( \Phi \) must have a spectral radius less than 1, where the spectral radius of a matrix is the absolute value of the matrix’s largest eigenvalue [43]. This is the same stability criteria used for the closed loop state space representations of discrete control systems [44].

In order to more closely examine the stability criteria outlined in Eq. (8), 1000 systems with different numbers of subsystems, design variables, and objective functions were randomly generated. All these systems were evaluated using the stability criteria outlined in Eq. (8), and the absolute value of their largest eigenvalue was found to be less than 1. These systems were then assigned random process architectures, similar to one of those shown in Fig. 2. The upper and lower bounds of the uniform distribution used to generate the design systems are shown in Table 1.

In order to reduce the number of possible parameters in the experiment that may bias the result, the values in the \( D \) matrix were set to zero to guarantee that all the design systems had equilibrium solutions at the origin. An equivalent approach could have been to shift the variables, moving all the system equilibriums to the origin. Each subsystem was given local control of one variable; the remaining variables were then randomly allocated to the different subsystems. Each system was then simulated until it converged to a solution or reached a maximum of 250 iterations. After a system reached 250 iterations, it was assumed to be a divergent system. A larger iteration limit could have been used, but these simulations were used to screen the initial set of 1000 systems and are summarized in Eqs. (9a) and (9b).

These divergent systems were examined in greater depth in Sec. 4.4. By studying how the design variable values change between iterations, a significant number of systems were identified as being divergent in spite of fulfilling the criterion in Eq. (8). This finding is significant because it was previously thought that the solution process architecture had no influence on the system stability [18]. This also experimentally demonstrates the first part of our hypothesis that the process architecture influences the stability of distributed design systems.

A critical aspect to assessing the process stability is the convergence criteria. Convergence is defined for these simulations to occur when all the design variables for a system have progressed to within 2% of their equilibrium value, as measured from the initial starting location. The 2% criterion is consistent with the linearization convergence criteria. Convergence is defined for these simulations to occur when all the design variables for a system have progressed to within 2% of their equilibrium value, as measured from the initial starting location. The 2% criterion is consistent with the linear system definition of settling time.

The initial value of all design variables was set to 1.0 to ensure that no system began at their equilibrium. For the systems modeled in this experiment, the stability can be determined independently from the starting location provided the system does not start at its Nash equilibrium. For these simulations, it was found that 115 of the 1000 systems, or 11.5% of them, were divergent based on the convergence criteria when simulated using a nonparallel architecture. An example of one of these systems for two process architectures is shown in Figs. 5 and 6. The system is simulated in Fig. 5 with a purely parallel architecture, while the system in Fig. 6 has a hybrid architecture.

For both Figs. 5 and 6, the number of design iterations is shown on the \( y \)-axis and the design variable values at each iteration are shown on the \( y \)-axis, where every design variable has a different color/shade and shape. These figures illustrate how the same distributed design system can have very different stability depending solely on the choice of process architecture.

The system shown in Figs. 5 and 6 has four unique subsystems and six unique design variables. Subsystem 4 controls three variables while each of the other subsystems controls a single variable. Simulation of the purely parallel solution process in Fig. 5 shows that all the design variables converge to the equilibrium solution at the origin from an initial starting location of 1.0 after the ninth iteration. The eigenvalues for this system were determined using the approach to assess stability in Refs. [36,39] and are summarized in Table 2, omitting eigenvalues with magnitude 0.

### Table 1 Convergence experiment parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. subsystems</td>
<td>4–10</td>
</tr>
<tr>
<td>No. DVs</td>
<td>4–15</td>
</tr>
<tr>
<td>( A )</td>
<td>−25 to 25</td>
</tr>
<tr>
<td>( C )</td>
<td>−25 to 25</td>
</tr>
</tbody>
</table>

### Table 2 System eigenvalues with current approach

<table>
<thead>
<tr>
<th>([1,2,3,4]) (Parallel) system eigenvalues</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eigenvalues</td>
</tr>
<tr>
<td>0.3153 ± 0.5574i</td>
</tr>
<tr>
<td>−0.2148 ± 0.4617i</td>
</tr>
<tr>
<td>0.3016</td>
</tr>
<tr>
<td>0.1006</td>
</tr>
</tbody>
</table>

---

**Fig. 5** Convergent—parallel solution process

**Fig. 6** Divergent—nonparallel solution process
Since the magnitude of the largest eigenvalue in Table 2 is less than 1, it is expected that this system’s Nash equilibrium is stable and the system will converge. In spite of this prediction, the system is unstable for the process architecture shown in Fig. 6. This reinforces our hypothesis that process architecture does influence the equilibrium stability. Examination of the assumptions inherent to discrete time linear systems provides qualitative insight into why the solution process architecture influences equilibrium stability. When a distributed design problem is modeled as a discrete time control system, it is assumed that [44]:

1. All subsystems update their design variable(s) simultaneously,
2. All the subsystems take the same time to update their design variable(s).

In a distributed design problem, the process of a subsystem solving for their design variables is analogous to sampling the states in a physical system. Models for discrete time systems using linear system theory require all samples be taken simultaneously (assumption one) and that the interval between samples be uniform and equal for all states (assumption two). When a distributed design system is arranged as a parallel solution process, like in Fig. 5, all subsystems solve their optimization problems simultaneously and assumption one is fulfilled. Each subsystem also waits for every other subsystem to finish solving their optimization problem and updating their design variables, before performing additional design iterations. This fulfills the second assumption and the stability of a parallel solution process architecture can be determined using Eq. (8).

When a design system is arranged in a nonparallel configuration, the design variables update at different, nonsimultaneous time for nonparallel elements. For example, for the process architecture in Fig. 6, subsystems 1 and 4 update simultaneously with respect to one another while subsystems 2 and 3 update at a different time. This difference means that the above approach can only be used to evaluate the stability of purely parallel process architectures.

Since it may not always be possible to arrange a distributed design system in a parallel configuration, it is important to be able to evaluate the system stability for different process architectures. A similar challenge has been faced for physical systems when it is not always possible to sample every state simultaneously [45,46]. An approach is presented in Sec. 4 that can convert a process architecture into an equivalent parallel system to determine its stability.

4 Parallel Equivalent Systems

In this paper, the term parallel equivalent is introduced and is defined as a system that has the same stability and transient response as a system arranged in a nonparallel configuration. Although a parallel equivalent behaves in the same manner as the nonparallel system it models, the RRS of each subsystem may not be the same in the parallel equivalent. This is because the information sharing relationships inherent to the process architecture are captured in the modified subsystem RRSs. Although the RRSs may be different, when complete design iterations are performed for a system and its parallel equivalent, identical design variable values and convergence behavior are achieved. The advantage to creating a parallel equivalent is that it can be well modeled by the update relationship outlined in Eq. (7), and linear system theory can be used to predict its behavior.

4.1 Parallel Equivalent Conversion Approach. The parallel equivalent approach is based on the recognition that the stability of a parallel system can be evaluated because it simultaneously updates all design variables. Examining the difference between the update relationships of the two process architectures in Figs. 5 and 6 demonstrates the impact the nonsimultaneous update of design variables can have on a solution process. Equation (10) shows a revised version of the design variable update relationship in Eq. (7) for the process architecture from Fig. 6 where subsystems 1 and 4 simultaneously update their design variables followed by subsystems 2 and 3 ([1,4]→[2,3]).

\[
\begin{align*}
\mathbf{x}_{1}(t+1) &= \Phi_{1} \mathbf{x}_{1}(t) + \Gamma_{1}, \\
\mathbf{x}_{2}(t+1) &= \Phi_{2} \mathbf{x}_{2}(t) + \Gamma_{2}, \\
\mathbf{x}_{3}(t+1) &= \Phi_{3} \mathbf{x}_{3}(t) + \Gamma_{3}, \\
\mathbf{x}_{4}(t+1) &= \Phi_{4} \mathbf{x}_{4}(t) + \Gamma_{4}, \\
\end{align*}
\]

(10)

In the first stage, subsystems 1 and 4 solve their optimization problem and update variables \(x_1\), \(x_4\), \(x_5\), and \(x_6\) using the appropriate rows from \(\Phi\). For example, \(\Phi_{1}\), indicates that the row associated with \(x_1\) is populated with the values in the first row of \(\Phi\). Note that although \(x_2\) and \(x_3\) are updated simultaneously, in the second stage, they depend on the updated values of \(x_1\), \(x_4\), \(x_5\), and \(x_6\). Because of this, the overall update relationship between states cannot be summarized in the form shown in Eq. (7), which is why the linear systems model of Sec. 3.4 cannot be applied to this nonparallel process architecture.

The approach we develop begins with the last stage of the nonparallel solution process, stage \(k\). We convert the final stage into an equivalent form at stage \((k−1)\). This continues until we rollback each stage into an equivalent parallel stage. When iterations require only a single stage, systems update in the same manner as parallel solution process architectures. To apply the proposed approach, three assumptions are made:

1. The Nash equilibrium is located at the origin,
2. The process architecture does not change for the entire solution process,
3. Each subsystem appears a single time in the specified architecture.

The first assumption mirrors similar assumptions made in Refs. [37,38] when evaluating the behavior of distributed design systems. Even when systems have a Nash equilibrium that is not located at the origin, a change in variable in Eq. (7) can be used to shift the system to the origin. It should be noted that although the stability of an equilibrium point is a function of the process architecture, the location of the equilibrium itself is not.

The second assumption requires that the architecture does not change for the entire solution process so that a state space representation can be used to model the process. If the process architecture changes, then the update relationship expressed in Eq. (7) cannot be used without modifying the system. The approach presented in this paper does not account for changing the process architecture during a design process. Creating adaptive process

<table>
<thead>
<tr>
<th>Table 3 Parallel equivalent process variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
</tr>
<tr>
<td>(k)</td>
</tr>
<tr>
<td>(d_{curr}^{\text{sync}})</td>
</tr>
<tr>
<td>(d_{sync})</td>
</tr>
<tr>
<td>(\Theta)</td>
</tr>
<tr>
<td>(\Phi_{peq})</td>
</tr>
</tbody>
</table>

101001-6 / Vol. 133, OCTOBER 2011 Transactions of the ASME
architectures that evolve as the design system converges, however, is an important area of future work.

Finally, applying the approach in this section requires that, for a single iteration of the complete process architecture, each design variable is only updated once. Fulfilling these requirements guarantees that the converted parallel equivalent architecture accurately models the system across all iterations. To facilitate the formulation and execution of the parallel equivalent process, the notation used is explained qualitatively in Table 3.

To demonstrate the steps used to create a parallel equivalent system, each step is explained and then demonstrated using an example distributed design problem. This example is the same system shown in Figs. 5 and 6 and its properties are discussed in Sec. 4.2.

4.2 Parallel Equivalent Example Problem. In this section, the four-subsystem problem with six design variables whose behavior is summarized in Sec. 3.4, is described in detail. The objective functions and local design variables of the system analyzed in both Secs. 4.3 and 4.4 are summarized in Table 4.

Since the vector of linear elements for this design system, \( \mathbf{D} \), was set equal to zero, it is omitted from the system summary. Each matrix in Table 4 contains the quadratic elements found in the \( \mathbf{A} \) and \( \mathbf{C} \) matrices for each design system, as denoted by Eq. (6), with the first row and column corresponding to \( x_1 \). The remainder of the rows and columns correspond to \( x_2 \) through \( x_6 \). The diagonal elements in each matrix correspond to the terms found in the \( \mathbf{A} \) matrix, while the remaining terms correspond to the cross terms from the \( \mathbf{C} \) matrix. The representation is upper diagonal with zeros populating the lower diagonal due to the commutative property of multiplication. The vector \( \mathbf{X} \) is populated by the design variables ordered from \( x_1 \) to \( x_6 \). For example, the objective function for subsystem 1 from Table 4 is written analytically in Eq. (11).

\[
F_1(\mathbf{X}) = 5.69x_1^2 - 0.51x_2^2 + 20.68x_3^2 - 24.47x_4^2 - 22.97x_5^2 \\
- 20.21x_6^2 - 4.76x_1x_2 + 14.87x_1x_3 + 10.97x_1x_4 \\
- 13.51x_1x_5 + 5.41x_1x_6 + 7.44x_2x_3 - 9.12x_2x_4 \\
+ 1.38x_2x_5 + 10.58x_2x_6 - 17.66x_3x_4 - 23.94x_3x_5 \\
+ 6.46x_3x_6 + 10.74x_4x_5 + 2.64x_4x_6 + 14.91x_5x_6
\]

Since subsystem 4 is the only subsystem controlling multiple design variables, the cross terms linking \( x_4 \), \( x_5 \), and \( x_6 \) to one another are equal to zero. This satisfies the requirement discussed in Sec. 3.4 that the RRSs for every design variable is updated independently. Using the objective functions from Table 4, the creation of the parallel equivalent is demonstrated for a convergent process architecture in Sec. 4.3.

<table>
<thead>
<tr>
<th>Subsystem 1: ( x_1 )</th>
<th>Subsystem 2: ( x_2 )</th>
<th>Subsystem 3: ( x_3 )</th>
<th>Subsystem 4: ( x_4, x_5, x_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( F_1(\mathbf{X}) = X )</td>
<td>( F_2(\mathbf{X}) = X )</td>
<td>( F_3(\mathbf{X}) = X )</td>
<td>( F_4(\mathbf{X}) = X )</td>
</tr>
<tr>
<td>(-5.69) &amp; (-4.76) &amp; (14.87) &amp; (10.97) &amp; (5.41) &amp; (-24.47) &amp; (10.74) &amp; (2.64) &amp; (-22.97) &amp; (-14.91) &amp; (-20.21)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-5.69) &amp; (-4.76) &amp; (14.87) &amp; (10.97) &amp; (5.41) &amp; (-24.47) &amp; (10.74) &amp; (2.64) &amp; (-22.97) &amp; (-14.91) &amp; (-20.21)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-6.22) &amp; (0.18) &amp; (2.17) &amp; (-10.83) &amp; (24.54) &amp; (-2.79)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-6.22) &amp; (0.18) &amp; (2.17) &amp; (-10.83) &amp; (24.54) &amp; (-2.79)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-6.22) &amp; (0.18) &amp; (2.17) &amp; (-10.83) &amp; (24.54) &amp; (-2.79)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-6.22) &amp; (0.18) &amp; (2.17) &amp; (-10.83) &amp; (24.54) &amp; (-2.79)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-16.93) &amp; (5.60) &amp; (11.50) &amp; (14.14) &amp; (3.42) &amp; (6.52)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-16.93) &amp; (5.60) &amp; (11.50) &amp; (14.14) &amp; (3.42) &amp; (6.52)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-16.93) &amp; (5.60) &amp; (11.50) &amp; (14.14) &amp; (3.42) &amp; (6.52)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-16.93) &amp; (5.60) &amp; (11.50) &amp; (14.14) &amp; (3.42) &amp; (6.52)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0) &amp; (0) &amp; (0) &amp; (0) &amp; (0) &amp; (0)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0) &amp; (0) &amp; (0) &amp; (0) &amp; (0) &amp; (0)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0) &amp; (0) &amp; (0) &amp; (0) &amp; (0) &amp; (0)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0) &amp; (0) &amp; (0) &amp; (0) &amp; (0) &amp; (0)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.3 Creation of a Parallel Equivalent. The solution process architecture chosen for this example is \([3]→[1,2]→[4]\). This architecture was determined to converge to its equilibrium solution after seven complete iterations when all design variables are initialized to one. The process used to create a parallel equivalent architecture is composed of three steps, which are outlined in Fig. 7.

The steps in Fig. 7 broadly describe the approach to create the parallel equivalents for process architectures with the forms discussed in Sec. 3.3 and shown in Fig. 2. Using the specified architecture and the data from Table 4, the first step in the parallel equivalent conversion process can be completed. All mathematical representations in this section have been truncated at four decimal places while calculations were carried to six decimal places during actual computations.

4.3.1 Step 1—Initialize Parameters \( \Phi, k, d_{\text{par}}, \) and \( d_{\text{sync}} \). In this step, the parameters required for the transformation of the system into a parallel equivalent are initialized. The first task in this step is the creation of a linear system model to represent the design system using the approach outlined in Ref. [36]. This model requires that the local design variables be decoupled and conversion to a parallel equivalent requires the Nash equilibrium be at the origin. Any system can be transformed to meet these requirements by performing a change in variables and shifting the Nash equilibrium, respectively. The system being analyzed already meets these criteria because the local design variables have been decoupled for each subsystem and the \( \mathbf{D} \) matrix is set equal to zero. The required tasks are, therefore, as follows.

![Fig. 7 Parallel equivalent conversion flow chart](image-url)
4.3.1.1 Populate the $\Phi$ matrix for the design system. The relationship used to populate the $\Phi$ matrix is described by Eq. (9a). This relationship essentially captures subsystem interactions that occur due to the coupling terms in the subsystem RRSs. A detailed description of this process can be found in Ref. [36]. The $\Phi$ matrix resulting from applying Eq. (9a) is summarized in Eq. (12). As expected, the diagonal entries of this matrix are zeros and the last three rows have no cross dependence on one another. We use the $\Phi$ matrix as formulated in Eq. (9a) as the starting point for the conversion to $\Phi_{eq}$ because it quantifies the coupling between subsystems for a known parallel process architecture.

$$\Phi = \begin{bmatrix} 0 & 0.1517 & -0.4736 & -0.3496 & 0.4303 & -0.1725 \\ 0.3440 & 0 & -0.1758 & -0.4667 & 0.5329 & -0.9965 \\ -0.0444 & 0.3189 & 0 & -0.1604 & -0.0565 & 0.3335 \\ -1.2612 & 1.1862 & 0.3322 & 0 & 0 & 0 \\ -0.0709 & 0.2058 & -0.1437 & 0 & 0 & 0 \\ 0.2320 & 0.2372 & -0.2096 & 0 & 0 & 0 \end{bmatrix}$$

$$d_{curr} = [4, 5, 6]$$

4.3.1.2 Set $k$ equal to the number of steps in the process architecture. Since there are three distinct stages in the [3]—[1, 2]—[4] process architecture, $k$ is set equal to 3. The $k$ value is used to track the number of stages in the current process architecture and acts as an index for large design problems with complex process architectures.

4.3.1.3 Record the indices of the DVs controlled by the subsystems in the $k$th step in $d_{curr}$. Given $k$ equal to three, the subsystem in the third stage is subsystem 4. Subsystem 4 controls $x_4, x_5,$ and $x_6$. The index of these design variables is defined to be their row in the $\Phi$ matrix. The set of design variable indices is therefore, $d_{curr} = [4, 5, 6]$. At this point, these are the only design variables that are synchronized because they are all determined simultaneously. The following step begins synchronizing the design variables from previous stages, as illustrated in Fig. 8.

In Fig. 8, the original process architecture is shown by the elements with a solid border. The first synchronization, with subsystem 4 being synchronized with subsystems 1 and 2, is shown by the dashed lines while the second synchronization is shown by the dotted lines.

4.3.1.4 Record the indices of the DVs controlled by the subsystems in the ($k-1$)th stage in $d_{sync}$. There are two subsystems in the second stage of this particular architecture, subsystem 1 and 2, which control design variables $x_1$ and $x_2$, respectively. The set of design variable indices composing $d_{sync}$ is therefore $[1, 2]$. This set represents the variables with which $d_{curr}$ must be synchronized. Since the synchronization process is iterative, only the result for the first synchronization, which synchronizes subsystem 4 with subsystems 1 and 2, is shown in detail.

4.3.2 Step 2—Populate $\Theta$ and Update $\Phi$. In this step, the subsystems in the $k$th stage are synchronized with those in the $(k-1)$th stage and the process by which the parallel equivalent is constructed is described. The basis for this step is the relationship used to populate the $\Phi$ matrix. This relationship essentially captures subsystem interactions that occur due to the coupling terms in the subsystem $H$s. To modify these RRSs, a matrix, $\Theta$, is constructed based on the design variables in $d_{sync}$ and used to transform the rows of $\Phi$ specified by $d_{curr}$. The transformation of $\Phi$ synchronizes the design variables in $d_{curr}$ with those in $d_{sync}$. When all the rows of $\Phi$ specified by $d_{curr}$ have been updated, the subsystems in the $k$th stage can be arranged in parallel with the subsystems in the $(k-1)$th stage, as shown in Fig. 8. The specific process used to populate the $\Theta$ matrix and update $\Phi$ is described in the remainder of this step.

4.3.2.1 Create an identity matrix $\Theta$ with the same dimensionality as $\Phi$ and update using Eq. (13). Since there are six unique design variables in this design problem, the $\Theta$ matrix in this case is a $6 \times 6$ identity matrix. The $\Theta$ matrix is used to synchronize the $k$th and $(k-1)$th stages of the process architecture. Starting with an identity matrix, $\Theta$ is populated based on the entries in the set $d_{sync}$ using Eq. (13).

$$\Theta = \frac{1}{|d_{sync}|} \sum_{i=1}^{m} \sum_{j=1}^{n} \left\{ \Theta(d_{sync}(i), j) + \Phi(d_{sync}(i), j) \right\} - 1, d_{sync}(i) = j$$

In Eq. (13), $|d_{sync}|$ is the number of entries in the set $d_{sync}$ or its cardinality. The notation $d_{sync}(i)$ refers to the value of the $i$th member of the set $d_{sync}$. The diagonal values of $\Theta$ are all unity to preserve the values of the design variables that are not being synchronized. For the variables in $d_{sync}$, however, the diagonal entries must be zero because a design variable cannot be coupled to itself. The other entries in the rows associated with variables in $d_{sync}$ are equal to their corresponding entry in $\Phi$ and adding the appropriate entry of $\Phi$ accomplishes this since $\Theta$ is initialized to zero for those entries.

The relationships in Eq. (13) are best understood using an example problem. For the problem in Figure 8, the set $d_{sync}$ is composed of two entries, 1 and 2, corresponding to the two variables to be synchronized, $x_1$ and $x_2$. For simplicity, only the final value, the $\Theta$ matrix, is shown in Eq. (14). The $\Theta$ matrix preserves the coefficients of the design variables not in $d_{sync}$ since they are multiplied by 1 and transforms the values linked to $d_{sync}$. The $\Theta$ matrix in Eq. (14) is used in the next step to update the appropriate entries of the $\Theta$ matrix.

$$\Theta = \begin{bmatrix} 0 & 0.1517 & -0.4736 & -0.3496 & 0.4303 & -0.1725 \\ 0.3440 & 0 & -0.1758 & -0.4667 & 0.5329 & -0.9965 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

4.3.2.2 Update $\Phi$ to $\Phi(d_{curr}) = \Phi(d_{curr}) \cdot \Theta$. Multiplying the rows of $\Phi$ associated with $d_{curr}$ by $\Theta$ synchronizes the design variables in $d_{curr}$ with those in $d_{sync}$. The resulting vectors are then substituted into the row in $\Phi$ associated with $d_{curr}$. The vectors resulting from this multiplication are shown in Eq. (15).
OCTOBER 2011, Vol. 133
Journal of Mechanical Design

ized, the next stage can be synchronized after updating the process using their own individual RRSs. With the two stages synchronized to the designers behavior and the designers themselves still operate lined because the matrix being constructed is a model for the Eq. (16). This does not violate the independence condition out-

\[
\Phi(4.)^T = \begin{bmatrix}
-1.2612 \\
1.1862 \\
0.3322 \\
0.4081 \\
-0.1913 \\
0.7210 \\
-0.1126 \\
-0.0984 \\
-0.9644
\end{bmatrix}
\]

\[
\Phi(5.)^T = \begin{bmatrix}
-0.0790 \\
0.2058 \\
-0.1437 \\
0.0708 \\
-0.0108 \\
-0.1463 \\
-0.0712 \\
-0.0792 \\
-0.1928
\end{bmatrix}
\]

\[
\Phi(6.)^T = \begin{bmatrix}
-0.2320 \\
0.2372 \\
-0.2096 \\
0.0816 \\
-0.0352 \\
-0.1414 \\
-0.0296 \\
-0.0266 \\
-0.1964
\end{bmatrix}
\]

In Eq. (15), the left hand vector is a row of \( \Phi \) and it is right multiplied by \( \Theta \). The resulting vector on the right is the new row for \( \Phi \) associated with the entry in \( d_{cut} \). In Eq. (16), the transformed \( \Phi \) with the new entries for rows 4, 5, and 6 (highlighted by the dashed line) is shown.

\[
\Phi = \begin{bmatrix}
0 & 0.1517 & -0.4736 & -0.3496 & 0.4303 & -0.1725 \\
0.3440 & 0 & -0.1758 & -0.4667 & 0.5329 & -0.9965 \\
0.4081 & 0.7210 & -0.1126 & -0.0984 & 0.9644 & 0.1928 \\
0.0708 & -0.0108 & -0.1463 & -0.0712 & -0.0792 & -0.1928 \\
0.0816 & -0.0352 & -0.1414 & -0.0296 & -0.0266 & -0.1964
\end{bmatrix}
\]

This overall process is repeated until all design variables are members of the set \( d_{cut} \), indicating that the system has been completely transformed into its parallel equivalent. The process outlined in steps 2 through 3 is repeated \((k-1)\) times and convergence is achieved when \( k = 1 \).

After all the stages have been synchronized, the new update matrix for \( \Phi \) that models the system as a parallel equivalent is complete. This matrix is shown in Eq. (17) along with its associated eigenvalues.

\[
\Phi_{peq} = \begin{bmatrix}
0.0210 & 0.0007 & 0 & -0.2377 & 0.4571 & -0.3305 \\
0.3518 & -0.0561 & 0 & -0.4385 & 0.5429 & -1.0551 \\
-0.0444 & 0.3189 & 0 & -0.1604 & -0.0565 & 0.3335 \\
0.3761 & 0.0386 & 0 & -0.2283 & 0.0487 & -0.7240 \\
0.0773 & -0.0574 & 0 & -0.0478 & 0.0874 & -0.2416 \\
0.0879 & -0.0803 & 0 & -0.0069 & 0.0346 & -0.2434
\end{bmatrix}
\]

\[
\lambda = [0.3819 -0.0187 \pm 0.2783j]
\]

The resulting parallel equivalent representation, \( \Phi_{peq} \) in Eq. (17) is much different from the original \( \Phi \) for the system. The most notable characteristic of \( \Phi_{peq} \) is that the third column is composed entirely of zeros, which suggest all the subsystem RRSs are independent of design variable \( x_3 \). This independence occurs because the subsystem controlling design variable \( x_3 \) is the first to solve its optimization problem. Since the value of \( x_3 \) is a function of all the other design variables, it is solved in terms of the other design variables in the first stage of the process architecture. If \( x_3 \) was dependent on its value from the previous iteration, then it would appear in the \( \Phi_{peq} \) matrix. In the context of the update relationship in Eq. (7), having zeros populate the third column of \( \Phi_{peq} \) can be qualitatively understood as the solution process being independent of the initial value for \( x_3 \). This makes sense because the initial value of \( x_3 \) is updated by subsystem 3 before any other subsystem has a chance to perform an optimization using \( x_3 \)‘s initial value. It is important to recognize that in spite of this, the distributed process still incorporates the input from subsystem 3 in the first stage within the entries for the other design variables.

Another significant property of the expression in Eq. (17) is that its eigenvalues are different from those for the parallel process architecture shown in Table 2. Since the spectral radius of the matrix in Eq. (17) is less than 1.0, this process architecture should have a stable Nash equilibrium. The stability of the system, along with the validity of the parallel equivalent transformation, is examined by plotting the system’s design variable values in Fig. 9 from a starting location at 1.0.

In Fig. 9, each of the design variables is graphed separately for clarity and the iteration number is plotted on the x-axis while the design variable value is plotted on the y-axis. For each graph in Fig. 9, two distinct data sets are plotted with respect to the iteration number, one as a solid staircase plot and the other as discrete symbols. The staircase plot for each graph was generated by transforming the \( \Phi_{peq} \) matrix shown in Eq. (17) into a discrete time state space model using the seq(\( \Phi_{peq} \), \( \Gamma \), \( C \), \( D \), \( t_0 \) ) function from the MATLAB control toolbox. This function creates a state space model for the system using the system parameters \( \Phi_{peq} \) and \( \Gamma \), already defined for this system, while \( C \) is a 1 x 6 vector with 1 as entry for the design variable being displayed and zeros elsewhere. The \( D \) value is a scalar and was set to zero while the \( t_0 \) entry is the sampling time for the system. Since typical time measures do not have a significant meaning for this investigation, it was set equal to 1 s, which corresponds to one sample per iteration. The second data set plotted in Fig. 9, denoted by the discrete symbols, was generated by simulating the system, allowing the subsystems to iterate using the specified process architecture until they converged to a solution. The results from this simulation accurately represent...
the discrete state space results generated using the $\Phi_{peq}$ matrix approach. The same approach applied to convergent design architectures can be applied to architectures that are not convergent.

4.4 Divergent Parallel Equivalent. In this section, the process architecture shown in Fig. 6 is revisited to examine the eigenvalues of its parallel equivalent. This process architecture is [1,4]→[2,3], and simulation showed it had an unstable Nash equilibrium. To analyze the eigenvalues of this process architecture, the approach outlined in Sec. 4.3 is used to create a parallel equivalent representation of the system, shown in Eq. (18).

$$
\Phi_{peq} = \begin{bmatrix}
0 & 0.1517 & -0.4763 & -0.3496 & 0.4303 & -0.1725 \\
0.7820 & -0.6281 & -0.3615 & -0.1203 & 0.1481 & -0.0594 \\
0.1289 & 0.1894 & -0.0940 & 0.0155 & -0.0191 & 0.0077 \\
-1.2612 & 1.1862 & 0.3322 & 0 & 0 & 0 \\
-0.0709 & 0.2058 & -0.1437 & 0 & 0 & 0 \\
-0.2320 & 0.2372 & -0.2096 & 0 & 0 & 0
\end{bmatrix}
$$

(18)

Examining Eq. (18), the vectors describing the behavior of $x_1$, $x_4$, $x_5$, and $x_6$ are identical to the original $\Phi$ matrix entries. This is expected since those DVs are controlled by subsystems 1 and 4 and already depend on $x^2$ in their update relationship. The other two DVs have been transformed in rows 2 and 3 of $\Phi_{peq}$. Similar to the representation in Eq. (16), this results in nonzero diagonal entries, which does not violate any of the assumptions for the system because $\Phi_{peq}$ is a model for the subsystem behavior and not the actual subsystem RRSs. The eigenvalues for $\Phi_{peq}$ in Eq. (17) are summarized in Table 5.

Since the magnitude of the largest eigenvalue is greater than 1.0, this analysis predicts the system is unstable. This prediction is validated by the simulation shown in Fig. 6. In addition to accurately predicting the stability of divergent distributed design processes, the parallel equivalent transformation can be used to model the discrete solution steps of these process architectures. The design variable value for $x_1$ is plotted in Fig. 10 with respect to the iteration number using the same MATLAB formulation as Sec. 4.4.

In Fig. 10, the staircase plot is again the design variable value as determined using the $\Phi_{peq}$ in Eq. (18). The symbols in the plot are the values for $x_1$ as determined through simulation. Similar to the plot of the design variables for the convergent system, the parallel equivalent representation correctly captures the values for $x_1$. Although not shown, the behavior of the other five design variables is also accurately reproduced.

Further validation of this approach is achieved by analyzing the 1000 quadratic distributed design problems used in Sec. 3.4. All these systems were originally defined as stable through examination of the eigenvalues of their unmodified $\Phi$ matrix. For each design system, the spectral radius of the resulting $\Phi_{peq}$ is determined to evaluate the system stability using the criteria in Eq. (8).

The parallel equivalent approach identified 99 design systems with spectral a spectral radius greater than or equal to 1, which corresponds to 99 unstable systems. Simulation of the same 1000 systems indicated that 115 of these systems were unstable. This set included all 99 identified by the parallel equivalent approach, as well as 16 additional systems without spectral radii greater than 1 for their $\Phi_{peq}$ representation. One limiting factor for the simulation of the systems was an upper bound of 250 iterations before the problem was assumed to be divergent. Relaxation of this limit for these 16 systems demonstrated that they successfully converged to a stable Nash equilibrium in finite time. These results validate the applicability of the parallel equivalent conversion process for distributed design systems with quadratic objective functions. Expanding this method to address systems with nonlinear RRSs is a topic of future work in Sec. 5.

5 Conclusions

In this paper, the assumption that the stability of distributed design systems is independent of the process architecture is examined. We propose the hypothesis that the process architecture chosen impacts the system stability by changing the aggregate system eigenvalues. The first part of our hypothesis is demonstrated by example; a distributed design system is shown to converge using one architecture choice and diverge using a different architecture. It is also demonstrated that there is a...
significant portion of distributed design systems, 99 of the 1000 simulated, which were previously thought to be stable across all process architectures that have at least one unstable process architecture.

The limitation of current approaches to determine system stability is shown to be based on the hidden assumption that all states, the design variables in distributed design, are updated simultaneously. This assumption is shown only to be true for parallel process architectures and the difference in the update relationship is shown for an example and an approach to synchronize the design variables to update simultaneously is proposed. This approach creates a parallel equivalent representation of a nonparallel process architecture that can be processed using digital control theory to assess system stability and model solution iterations. The proposed approach adds three assumptions to those inherent to noncooperative distributed design formulations, which are as follows: (1) The Nash equilibrium is located at the origin, (2) the process architecture does not change, and (3) each subsystem appears once per iterative loop. Further, closed form representations of the subsystems’ RRSs are required for implementing the approach.

We experimentally validate the proposed approach and demonstrate by example the second part of our hypothesis, that process architecture choices change the system eigenvalues. A topic of our current work is to mathematically prove in general that the architecture influences the system eigenvalues and therefore the stability of the process.

Determining the stability of a system has broader impacts in any scenario where decision makers are members of large decentralized decision networks. Stable process architectures can be counted on to converge and require limited oversight to guide them to equilibrium solutions. Before a decision process begins, stakeholders are guaranteed that they will eventually reach agreement. Further, process architectures that would result in unstable or marginally stable systems can be avoided to minimize time spent in a design or development process before corrective action is taken.

One of the limitations of the proposed approach is that it assumes that the subsystem’s rational reaction sets are linear, which restricts its applicability to systems with quadratic objective functions. Many systems can be well modeled using a quadratic representation, but there are a significant number of systems with nonlinear reaction sets. Future work will address the challenge of assessing the stability of nonlinear systems and extending the parallel equivalent representation to capture the influence of process architecture on these systems.

In attempting to expand the applicability of the proposed approach, another challenge is to develop techniques to appropriately assess the overall stability of all process architectures for an individual system. Knowing which process architectures are convergent or divergent provides insight to design managers to identify favorable system decompositions. Another important expansion of the current approach is to incorporate more diverse process architectures that enable overlapping and nested sequential tasks.

Another challenge is to utilize the eigenvalue information to effectively identify process architectures with the desired convergence properties. These may be systems that converge as quickly as possible to an equilibrium or those that converge with the least amount of design variable oscillation. With an effective model for any individual process architecture, it becomes possible to investigate these system properties in order to establish stable architectures that also possess the convergence properties desired for the overall design system.

Acknowledgment

We would like to thank the National Science Foundation, Grant No. DMII-0322783, and the Moog Corporation for their support of this work.

References


