Parallel, Multistage Model for Enterprise System Planning and Design

Harrison M. Kim, Shen Lu, Jin Suk Kim, and Byoung-Do Kim

***Abstract*—This paper describes a parallel, multistage optimization approach for enterprise system design and planning where the design of a system is linked with its planing and operations (resource allocation). Our approach is composed of two parts: a multistage formulation and a task-parallel algorithm. The formulation utilizes the quasi-separability of the multistage decision making structure, i.e., allowing relaxation by defining the linking variables for adjacent stages of decision making. The task-parallel algorithm enables optimal load balancing of the tasks, and it is validated in the demonstration case where an airline plans to introduce multiple new aircraft to capture dynamically changing travel demand. A linearly increasing computational load is assumed as the number of stages increases due to the complexity added onto the upcoming future stages in the optimization processes. The proposed task parallel algorithm demonstrates significant speedups and parallel performances by utilizing this linearity.**

***Index Terms*—Design, optimization, parallel enterprise system, planning.**

I. Introduction

ENTERPRISE systems include a variety of elements such as design, manufacturing, planning, consumer use, operations, take-back, recovery, etc. Traditionally, system design decisions have been separate from system operations or usage decisions; however, conventional artifact-oriented system design is gradually transformed into service-oriented system design under the enterprise system notion. In this paper, the term "enterprise system" is defined as a holistic system that encompasses multiple, distinct system domains. Thus, enterprise system characteristics are influenced by engineering design, marketing, planning, sales, etc. as they relate to a firm's business decisions such as maximize profit, minimize cost, and meet the performance goals.

It is not critical to plan for next generations of new products in the case of artifact-oriented product design with no service component. However, the enterprise must be able to plan ahead to maximize its profit regarding next generations of products and services with an increased importance of the service component of products. Critical decisions in planning include product take-back timing, component reuse/recycle/remanufacture, product portfolio management, service portfolio management, etc. In the case of green product portfolio design, for example, a subset of components from previous generation products may be used again to reduce cost and minimize a negative environmental impact. The enterprise often needs to consider multiple generations of product portfolios with service components taken into account simultaneously. The design and planning model becomes very complex as a result.

This simultaneous decision making often involves discrete/integer decision variables in the problem. Thus, the multigeneration (or, multistage) system planning and design problem becomes intractable from the mathematical point of view resulting in a mixed integer nonlinear programming problem (MINLP). Previous works show this MINLP nature in many different application areas. Crossley and Mane [1] integrated resource allocation and system design under the notion of system of systems and solved them utilizing discrete optimization methods. Kim and Hidalgo [2], [3] extended this idea to a dynamic environment in which multistage aircraft design and operation is formulated and solved as a multilevel, multidisciplinary problem utilizing methods such as analytical target cascading [4].

The increasingly complex engineering application has been one of the major challenges faced by the design, as well as computing communities. With the emerging notion of enterprise in engineering design and system management, one effective way to meet this challenge is to utilize the theory and tools from concurrent engineering and parallel computing. Concurrent engineering and parallel computing has been implemented in engineering design and its optimization in various aspects. The notion of concurrent engineering (CE) approaches has been traditionally applied to address the process downstream aspect of simultaneous (multidisciplinary) design decision making [5]–[8]; and, for the optimization aspect, major efforts have been focused on the concept of parallel optimization (PO) at different levels [9].

Most of the CE applications fall into two categories. The first category emphasizes incorporating a number of different life-cycle perspectives relevant to a product at the product design stage, e.g., manufacturing, assembly, reuse/recycling [10]. Another category focuses on the computer-aided collaborative engineering design, and many computational models have been proposed to facilitate this application [11], [12].

As for the optimization aspect, major efforts have been organized under the concept of PO and parallel processing (PP). The multidisciplinary PO applications in a system-subsystem structure can be categorized into three levels: 1) subsystem analysis; 2) subsystem optimization; and 3) system optimization and coordination.

Manuscript received February 16, 2009; revised August 21, 2009; accepted August 21, 2009. Date of publication February 08, 2010; date of current version April 07, 2010.

H. M. Kim and S. Lu are with the Department of Industrial and Enterprise Systems Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA (e-mail: hmkim@illinois.edu; shenlu2@illinois.edu).

J. S. Kim is with the Department of Computer Science, University of Seoul, Seoul, Korea (E-mail: jskim@venus.uos.ac.kr).

B.-D. Kim is with the Texas Advanced Computing Center, Austin, TX 78758 USA (e-mail: bdkim@tacc.utexas.edu).

Digital Object Identifier 10.1109/JSYST.2009.2039733

At the subsystem analysis analysis level, PP techniques, such as parallel algorithms for systems of algebraic equations, domain decomposition techniques, and so on, are applied to disciplinary analysis process to achieve computational speedup [13]. The subsystem level PO includes parallel implementation of various optimization algorithms, including mathematical programming algorithms [14], [15] as well as global optimization algorithms [16]–[18]. PO of single disciplinary problems also falls into this category.

At the system optimization and coordination level, efforts have been made to address coordination among the subsystems. Many multidisciplinary design optimization (MDO) approaches, such as Bi-Level Integrated System Synthesis [19], Concurrent Subspace Optimization [20], Collaborative Optimization [21], and Analytical Target Cascading [22], are implemented such that subsystem optimization processes are executed in parallel. Some of these approaches have been proven to generate solution sequences convergent to a local minimizer of the problem [23]. Furthermore, MDO environments have been proposed to assist multidisciplinary design decision making [24], [25].

It is well known generally that the allocation of arbitrary tasks onto a system of processors is an NP-complete problem. An approximation or heuristic algorithm is sometimes pursued as a practical alternative due to this complexity. The heuristic algorithms for task allocation are classified into two categories: *static* and *dynamic*. In static algorithms, the task allocation is determined off-line before the tasks are executed [26], [27], while dynamic algorithms determines the task allocation at runtime (online) [28]. There is no guarantee on the performance of a heuristic algorithm on a general problem in principle. Therefore, some understanding of a problem's characteristics may be desirable while developing heuristics for the problem. In this paper, we focus on static task allocation in the scenario where a set of tasks is not interdependent, and their processing time is a linear function of their index in the task set.

This paper presents two main contributions motivated by: 1) engineering system design under the enterprise notion and 2) parallel computing capability to overcome complexity. First, a multistage formulation is presented by integrating multilevel and multistage optimization utilizing the quasi-separability; then an efficient parallel coordination algorithm is presented to utilize the quasi-separable structure fully. The multistage decision making for design and operations of a system under the enterprise notion is modeled specifically in a manner that is suitable for a parallel computing paradigm. For the scenario where the processing time of single stage decision problem is a linear function of its stage index, the proposed parallel coordination algorithm ensures the optimal task parallelism for the maximum utilization of computing resources while allowing decision making autonomy in the enterprise.

The rest of the paper is organized as follows. Section II describes the pseudohierarchical multistage formulation, followed by multistage coordination based on the parallel task-decomposition algorithm. An illustrative example where an airline orders multiple, customized future aircraft demonstrates and validates the proposed approach.

II. Parallel Enterprise System Planning and Design

A. Problem Statement

The problem addressed in this paper concerns two aspects of enterprise system planning and design: 1) how to model a multigeneration system (or product) design when the service/operation component must be considered in system design phase and 2) how to utilize parallel computing capability in an optimal manner when the number of processors is smaller than the number of tasks (i.e., limited computing resources). Specifically, the problem description can handle the collaborative design and planning problems such as manufacturing machine design and planning; supercomputing system acquisition decision making for optimal allocation, etc. The all-in-one (AIO) approach (i.e., solving in one large problem) is not a practical option due to its multistage nature. Thus the decomposition-based approach must be introduced to model the decision making. Further, as the number of stages is increased, finding an optimal set of solutions becomes a challenge under limited computing resources. To overcome this challenge, a new coordination algorithm is needed for an efficient use of multiple, autonomous computing resources (or processors). The AIO formulation and the decomposed, individual stage formulation linked between stages are presented in the next two subsections, respectively, to show the equivalence between the two.

B. The Original All-in-One Formulation

The original AIO multistage problem is stated as follows. Given system targets (i.e., performance goals) $\mathbf{T}_1, \ldots, \mathbf{T}_p$ for all stages 1 through p, the enterprise system generates responses (i.e., output) $\mathbf{R}_1, \ldots, \mathbf{R}_p$ subject to design and operational constraints $\mathbf{g}$ (only inequality constraints are considered here for simplicity)

$$\begin{aligned} P_{AIO}: \quad & \min f(\mathbf{R}_1, \ldots, \mathbf{R}_p; \mathbf{T}_1, \ldots, \mathbf{T}_p) \\ \text{w. r. t.} \quad & \mathbf{x}_1, \ldots, \mathbf{x}_p \\ & \mathbf{y}_{1,2}, \mathbf{y}_{2,1}, \mathbf{y}_{2,3}, \ldots, \mathbf{y}_{p,(p-1)} \\ \text{s. t.} \quad & \mathbf{R}_1 = \mathbf{R}_1(\mathbf{x}_1, \mathbf{y}_{1,2}), \\ & \mathbf{R}_2 = \mathbf{R}_2(\mathbf{x}_2, \mathbf{y}_{2,1}, \mathbf{y}_{2,3}), \ldots, \\ & \mathbf{R}_p = \mathbf{R}_p(\mathbf{x}_p, \mathbf{y}_{p,(p-1)}) \\ & \mathbf{g}_1(\mathbf{x}_1, \mathbf{y}_{1,2}) \leq 0, \\ & \mathbf{g}_2(\mathbf{x}_2, \mathbf{y}_{2,1}, \mathbf{y}_{2,3}) \leq 0, \ldots, \\ & \mathbf{g}_p(\mathbf{x}_p, \mathbf{y}_{p,(p-1)}) \leq 0. \end{aligned} \tag{1}$$

To evaluate each response and constraint function, local design variables that belong to a single stage i, denoted as $\mathbf{x}_i$, and linking design variables shared by the current stage i and its adjacent stages $(i-1)$ and $(i+1)$, denoted as $\mathbf{y}_{i,(i-1)}$ $\mathbf{y}_{i,(i+1)}$, are taken as inputs. Note that the linking variables are only defined for adjacent stages shown in subscripts $(i, i-1)$ and $(i, i+1)$. At the optimum, while the objective is minimized, the

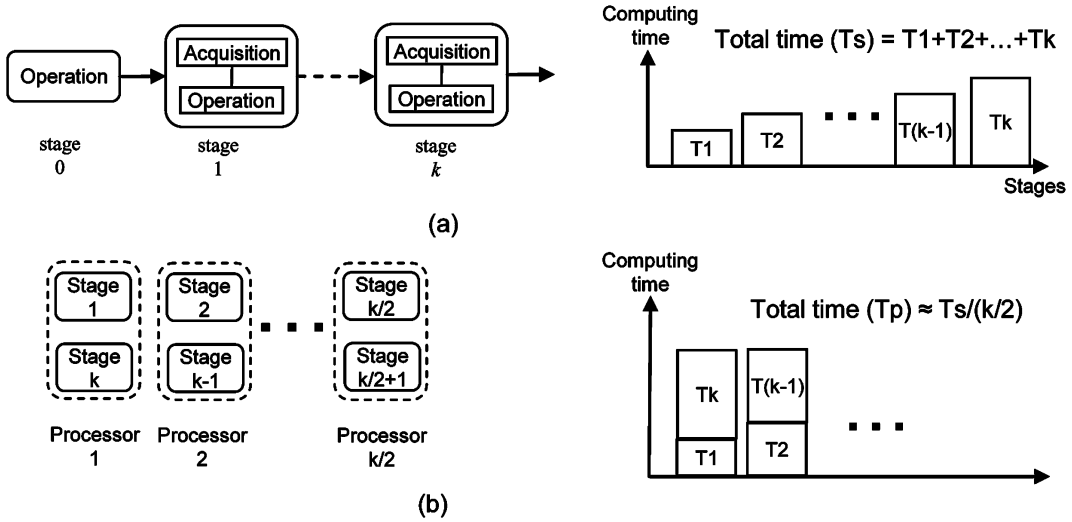


Fig. 1. Parallel multistage optimization model. (a) Sequential approach. (b) Parallel approach.

linking variables must converge to the same values, for example, $\mathbf{y}_{(i-1),i} = \mathbf{y}_{i,(i-1)}$ at the stages $(i-1)$, i.

This adjacent sequencing of linking variables implies that design decision making at the current stage is directly influenced by the decision making at the immediate previous stage, and it will affect the immediate following stage. For some multistage design problems, this sequencing is straightforward; for example, structural design precedes fatigue design in structural optimization. For other problems, however, the sequencing is not straightforward; for example, unmanned aerial vehicle (UAV) design for surveillance mission vs. attack mission in one operating system. The key assumption in this paper is that the system operations are represented as stages defined in a sequential manner as shown in Fig. 1.

C. Individual Stage Formulation

The multistage framework integrates the individual stage multidisciplinary design optimization in the horizontal (i.e., stages) direction. Traditional multilevel formulation decomposes a single overall problem into multiple subproblems at multiple levels, and traditional multistage problems formulate the sequential decision making as a series of smaller size subproblems along the time (or stage) horizon. By combining these two paradigms, first, at each stage, an integrated design and operations optimization problem (usually a mixed integer nonlinear programming (MINLP) problem) is formulated and solved. Then, these separate MINLP problem solutions communicate with each other to reach a consensus (i.e., matching values) for the links between different stages (Fig. 1).

Before proceeding with individual stage formulation, it is assumed that the objective function in the all-in-one problem (1) is separable without loss of generality due to the multistage notion of the current framework ($p = \text{number of stages}$)

$$f(\mathbf{R}_1, \ldots, \mathbf{R}_p; \mathbf{T}_1, \ldots, \mathbf{T}_p) = \sum_{i=1}^{p} f(\mathbf{R}_i, \mathbf{T}_i) = \sum_{i=1}^{p} ||\mathbf{R}_i - \mathbf{T}_i||_2^2 \quad (2)$$

where the squared L-2 norm $||\cdot||_2^2$ is used to calculate the deviation between the responses and targets. Note that some MINLP solvers rely on the continuous differentiability of the problem's continuous relaxation. Therefore the use of the squared L-2 norm may be critical, depending on the MINLP solver.

Based on the separability of the objective, at stage i, the individual system design problem is written as follows:

$$\begin{aligned} P_i: \quad & \min ||\mathbf{R}_i - \mathbf{T}_i||_2^2 + \epsilon_i \\ \text{w. r. t.} \quad & \mathbf{x}_i, \mathbf{y}_{i,(i-1)}, \mathbf{y}_{i,(i+1)}, \boldsymbol{\epsilon}_i \\ \text{s. t.} \quad & \mathbf{R}_i = \mathbf{R}_i(\mathbf{x}_i, \mathbf{y}_{i,(i-1)}, \mathbf{y}_{i,(i+1)}) \\ & \mathbf{g}_i(\mathbf{x}_i, \mathbf{y}_{i,(i-1)}, \mathbf{y}_{i,(i+1)}) \le 0 \\ & ||\mathbf{y}_{i,(i-1)} - \mathbf{y}_{(i-1),i}||_2^2 + ||\mathbf{y}_{i,(i+1)} - \mathbf{y}_{(i+1),i}||_2^2 \\ & \le \epsilon_i. \end{aligned} \quad (3)$$

As described in the problem statement, for example, new systems are added at each stage to the existing systems (adding to the existing fleet), thus increasing the scale of the individual optimization problems described in (3). This trend is addressed in Section III.

III. Task-Parallel Algorithms for Enterprise System Coordination

In this section, we describe the details of the coordination algorithm for task parallelism for the proposed multistage planning/design problem. First, the linearity assumption is introduced.

A. The Linearity Assumption

In the multistage optimization process, each stage is linked with adjacent stages, and the computational load becomes larger as the index of the stage increases due to increased complexity of the process. The most distinguishing characteristics of the current research are: 1) linearly increasing computational load for each stage [Fig. 1(a)] and 2) data communication for the design variables between stages involves only the beginning and

the end of individual system design and planning. **Task parallelism can be applied to this type of application by considering the stages as individual task process.** In principle, it is well known that finding the best possible allocation of arbitrary tasks onto a system of processors is a NP-complete problem. Numerous research works have been done with heuristic approaches in order to solve this type of problem. For example, [26] compares several scheduling heuristics for a class of tasks with no interdependencies. Additionally, [27] proposes an online heuristic scheduling algorithm for grid computing. In this paper, a parallel load-balancing algorithm is proposed for the parallel execution of the airline application based on its computational characteristics, i.e., linearity in the processing time of the multistage optimization process. The objective of the parallel algorithm is to assign task nodes to processors so as to minimize overall execution time of the application.

B. Coordination Algorithm Description

Suppose we are given a set of n independent tasks and are required to assign the tasks on to p processors. For the multistage airline application, let n be the number of stages of the optimization process (i.e., planning time horizon). Dividing n by p results in number m, which ideally represents the number of tasks assigned onto each processor if the computational load for each stage is identical. The reality with most scientific applications is not the case, however, as our model also has a varying computational load for each stage as discussed in previous section. The monotonic increase of the processing time (linearity as described in the previous subsection) with the multistage can be expressed as

$$\{T = T_i : i = 1, 2, \ldots, n | t(T_1) < t(T_2) < \cdots < t(T_n)\} \quad (4)$$

where $t(T_i)$ denotes the processing time of task at i_{th} stage with a set n of tasks. This time does not include the possible communication time for input data transfer and data transfer between the stages, which is assumed to be negligible compared to individual stage MINLP problem solving.

With n number of tasks (optimization process stages), the total processing time is partitioned into m number of cells, which has p number of processing tasks. Fig. 2 shows the notation of the partitioning cells and their elements. The C^i indicates i_{th} cell among the m number of cells, and c_j^i indicates j_{th} processing time in the i_{th} cell. Fig. 2 part a) shows the case where the m, the number of cells, is even numbers and Fig. 2 part b) shows the case where m is odd numbers. In case of the odd m, we utilize the notation of C^M, C^L and C^R in order to represent median cell, left hand side cell and right-hand side cell, respectively.

Another assumption for this application with the linearity of the processing time is that the number of task n is dividable by number of processors p without remainder, i.e., $\mathrm{mod}(n, p) = 0$. The case where the integer n is not dividable by p can be converted into the $\mathrm{mod}(n, p) = 0$ case by zero-load task padding. The latter case will be discussed later in this section.

The pseudocode given below presents the load-balancing algorithm with three different cases of task-processor mapping. The first case is where m is an even number (with either an even or odd number of p), and the second is the case where m is the odd number with an even number of p. An odd number of m with an even number of p is the third case. These three cases cover all the possible cases of $\mathrm{mod}(n, p) = 0$.

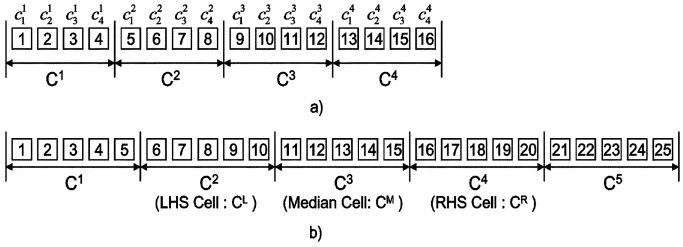


Fig. 2. Notation of the subcells and it elements. (a) Even number of m. (b) Odd number of m. (a) $n = 16, p = 4, m = 4$. (b) $n = 25, p = 5, m = 5$.

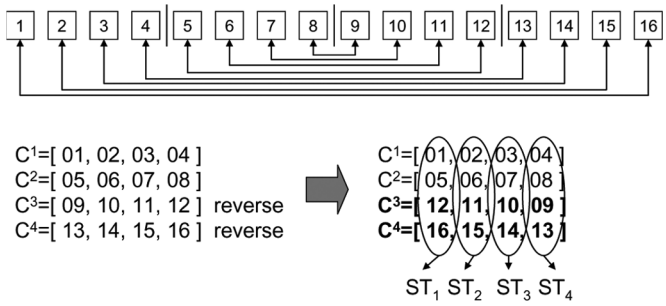


Fig. 3. Case I: Even number of m case with an example of $m = 4, p = 4$.

Algorithm Load-Balancing (T)

BEGIN

IF m **is an even number (Case I)**

for $i = (m/2) + 1$ **to** m **begin** $\mathrm{reverse}(C^i)$; **end**

ELSEIF (m **is an odd number) AND (**p **is an odd number) (Case II)**

$\mathrm{split}(C^{m+1/2})$; $\mathrm{subcell\_reverse\_o}(C^{(m+1/2)+1})$;

for $i = m + 1/2$ **to** m **begin** $\mathrm{reverse}(C^i)$; **end**

ELSE(Case III)

$\mathrm{rotation\_split}(C^{(m+1/2)})$; $\mathrm{subcell\_reverse\_e}(C^{(m+1/2+1)})$;

for $i = m + 1/2$ **to** m **begin** $\mathrm{reverse}(C^i)$; **end**

ENDIF

$ST_j \leftarrow \phi$; for $1 \le j \le p$

for $j = 1$ **to** p

for $i = 1$ **to** m

$ST_j \leftarrow ST_j \bigcup \left\{ c_i^j \right\}$

END

For Case I, we combine the p tasks into $(p/2)$ pairs with equal computational load. Since m is an even number, $(p/2)$ is dividable by p. Therefore, an even load balancing can be expected by assigning $(m/2)$ of these pairs to each processor. Technically,

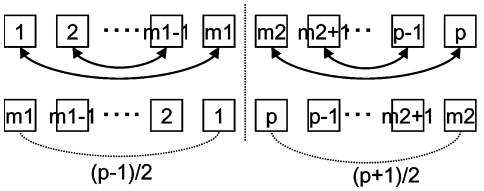


Fig. 4. Case II: subcell_reverse_o() function operation mechanism for odd number of m, odd number of p: $m_1 = ((p-1)/2), m_2 = ((p+1)/2)$.

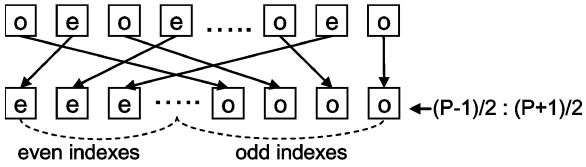


Fig. 5. split() function operation for the median cell (o represents odd number; e represents even number).

this can be implemented by a simple reverse function. The reverse() is a function that reverses the order of elements in a given cell. Fig. 3 shows the reverse mapping operation for the even number of m case with an example of $m = 4, p = 4$. The final subtask group ST_j gets a group of task elements for the optimal load balancing by taking j_{th} element of the cell.

The other two cases with odd m's are a bit more complicated than the first case. We combine the tasks in the three middle cells so that their computational load can be optimally allocated to the p processors; then the remaining $m-3$ cells are allocated in a manner similar to the Case I. Case II is where an odd number of elements p is in odd number of cells m. The mirror image mapping can be performed until the three cells in the middle, C^L, C^M and C^R, remain. The reverse mapping does not work for these three cells due to the asymmetry with the number of task elements. Here, we introduce two new functions; subcell_reverse_o() and split() that are applied to the right hand side cell, C^R, and the median cell, C^M, respectively. The operation of the subcell_reverse_o() function is shown in Fig. 4. As illustrated in the figure, the task elements in the C^R are partitioned into two groups at the ratio of $((p-1)/2) : ((p+1)/2)$, then the order of the task elements in each subcell is reversed by the reverse() operation.

Fig. 5 shows the split() function operation. In this operation, the task elements are being split into the same groups as in the subcell_reverse_o(), then the tasks at even number of indexes move to the first group, while the odd number indexed task elements move to the second group. Applying these operations to the right hand side and median cells, followed by the reverse() operation, results in an optimal load of $(t(T_n) \cdot t(T_{n+1})/2p)$ for the final subtask group ST_j.

Case III, with odd number of m and even number of p, goes though a similar operation as Case II. When it comes down to the operation for the three middle cells C^L, C^M and C^R, it requires one more step. Subcell_reverse_e() and rotation_split() functions are introduced in the pseudocode and are illustrated in Figs. 6 and 7 respectively. As shown in the Fig. 6, the subcell_reverse_e() does the same operation as the subcell_reverse() does, but it partitions the task elements into three groups rather than two. The grouping ratio is $1 : ((p-2)/2) : (p/2)$ and the first task element in the cell

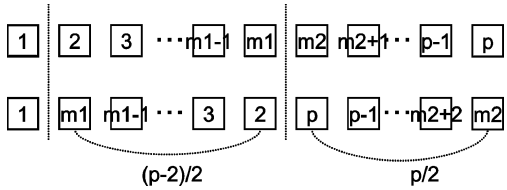


Fig. 6. Case III: Subcell_reverse_e() function operation for odd number of m, even number of p: $m_1 = (p/2), m_2 = (p/2)+1$.

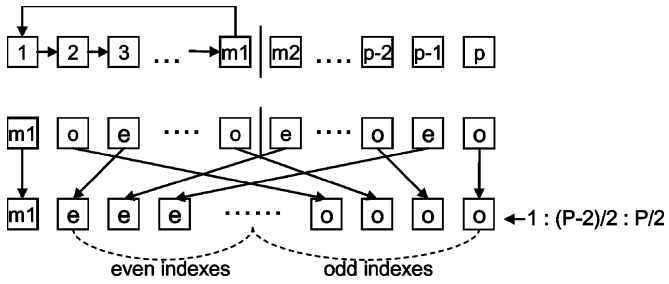


Fig. 7. Case III: Rotation_split function operation for odd number of m, even number of p: $m_1 = (p/2), m_2 = (p/2)+1$.

keeps the original position through the operation. By keeping the first index location, the rest becomes the same operation as the subcell_reverse() used in the case II.

The rotation_split() shown in Fig. 7 moves the $(p/2)$th task element to the first index location and moves the rest of tasks up one higher index position (rotation). After this additional step, the rest of the split operation is the same as the split() function in the case II but with $1 : ((p-2)/2) : (p/2)$ partitioning.

These special mapping operations in Case II and III result in a new set of task cells. From this point, the simple mirror image reverse mapping operation produces final subtask groups with optimal load balance.

Theorem 1: Assume that $T = \{T_i : i = 1, 2, \ldots, n | t(T_i) = a \cdot i + b; a, b : constant\}$ represents the processing time of a set of n tasks and that the number of processors p is a divisor of n. The load-balancing (T) algorithm maps the set of tasks with linearly increasing processing times on a p processor system with minimum makespan, where the makespan denotes the maximum total processing time on any processor.

Proof: The linearity of processing times $t(T_i) = a \cdot i + b$ can be simplified without loss of generality as follows: $t(T_i)$, i.e., $t(T) = 1, \ldots, n$. The lower bound for the makespan is $\lceil n(n+1)/2p \rceil$ because $\sum_{i=0}^{n} i = n(n+1)/2$. The cases described in this proof are the same cases in the algorithm in Section III-B.

Case I: The sum of the processing times in ST_j is

$$\sum_{i=1}^{m} c_j^i = \frac{m}{2} \cdot (1+n) \tag{5}$$

which is equal to the lower bound for the makespan.

Case II: The sum of the processing times in ST_j is

$$\sum_{i=1}^{m} c_j^i = \sum_{i=1}^{(m-3)/2} \left(c_j^i + c_j^{m-i+1}\right) + c_j^L + c_j^M + c_j^R$$
$$= \frac{(1+n)(m-3)}{2} + \left(c_j^L + c_j^R\right) + c_j^M$$

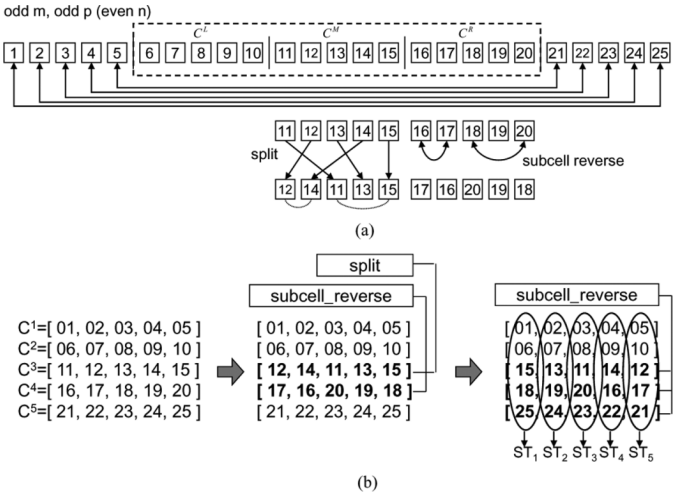


Fig. 8. Example problem of case II: $m = 5, p = 5, n = 25$. (a) Split and $\mathrm{subcell\_reverse\_o}()$ operation for C^M and C^R. (b) Task mapping steps of the load balancing process.

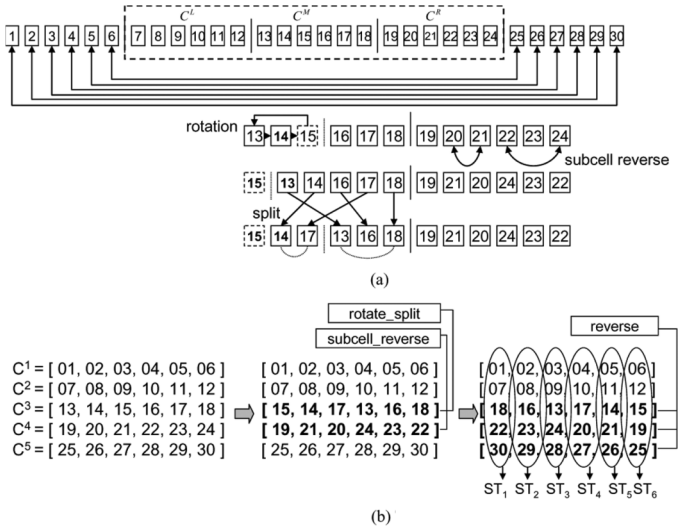


Fig. 9. Example problem of case III: $m = 5, p = 6, n = 30$. (a) Rotate_split() and $\mathrm{subcell\_reverse\_e}$ operations for C^M and C^R. (b) Task mapping steps of the load balancing process.

$$= \frac{(1+n)(m-3)}{2} + (1+n+\Delta_1) + \left(\frac{1+n}{2} - \Delta_1\right)$$
$$= \frac{m(n+1)}{2} \tag{6}$$

where $-(p-1/2) \le \Delta_1 \le (p-1/2)$.

Case III: The lower bound for the makespan is $(m(n+1)+1/2)$ because $m(n+1)$ is an odd number. The sum of the processing times in ST_j is as follows:

$$\sum_{i=1}^{m} c_j^i = \sum_{i=1}^{(m-3)/2} \left(c_j^i + c_j^{m-i+1}\right) + c_j^L + c_j^M + c_j^R$$
$$= \frac{(1+n)(m-3)}{2} + \left(c_j^L + c_j^R\right) + c_j^M$$
$$= \frac{(1+n)(m-3)}{2} + (1+n+\Delta_2) + \left(\frac{n+2}{2} - \Delta_2\right)$$
$$= \frac{m(n+1)+1}{2} \tag{7}$$

where $-(p-2/2) \le \Delta_1 \le (p-2/2)$. Therefore, the load balancing algorithm always finds minimum makespan for tasks-processors mapping.

Q.E.D.

We note that the stated optimality holds when the linearity assumption of task processing time is satisfied. If the task processing time is not linear with respect to the stage number, the allocation generated by the Load-Balancing(T) algorithm may not be optimal.

Two examples for the Case II and the Case III are presented in this section. For Case I, an example problem was shown in Fig. 3 with the algorithm description in the previous section. Fig. 8 demonstrates an example of Case II where $m = 5, p = 5$. The optimal load balance for each processor in this problem has value of 65. As shown in the figure, the right hand side cell C^R and the median cell C^M go through the $\mathrm{subcell\_reverse\_e}()$ and split() operations respectively as shown in part (a). After the special operations for the middle cells, the reverse() operation is applied resulting p number of subtask groups as shown in part (b). The computational load of each subtask group, which is sum of the task elements in the group, has the optimal value of 65.

An example problem of Case III where $m = 5$, $p = 6$ is shown in Fig. 9. The optimal load balance for each processor in this problem has a value of 77.5. The $\mathrm{subcell\_reverse\_e}()$ and $\mathrm{rotation\_split}()$ operation are applied for the C^R and C^M, respectively, as shown in Fig. 9(a). The final subtask groups ST_j have m number of task elements where the sum of the tasks is either 77 or 78, which satisfies the lower bound of the optimal value 78, as shown in Fig. 9(b).

So far, we have investigated cases where $\mathrm{mod}(n,p) = 0$ for the load balancing algorithm. The cases of $\mathrm{mod}(n,p) \neq 0$ can be converted into the $\mathrm{mod}(n,p) = 0$ problem by adding a minimum number of zero-load tasks in front of the original tasks. Here the minimum number means the smallest number of additional tasks that makes the $\mathrm{mod}(n,p) \neq 0$ to be one of the $\mathrm{mod}(n,p) = 0$ cases. Once the zero-load tasks padding is done, the problem falls into one of three categories that can be dealt with by the load-balancing algorithm without violating the assumption of the linear increase of the computational load.

IV. Demonstration Case

An airline fleet management and planning case is considered in order to demonstrate the multistage formulation and parallel coordination algorithm. Eight new aircraft are to be added to the existing fleet, one after another in each of the eight upcoming stages (e.g., every year), to accommodate dynamically changing travel demand. Complexity of the problem is increased significantly due to the multistage aspect of planning, although the route configuration in Fig. 10 seems simple.

The airline intends to plan ahead as to what type of aircraft should be introduced in the future stages before ordering (or customizing) them. The airline's objective is to minimize its direct operating costs (DOC) throughout the eight future stages [29]. Initially (the zero stage), the airline has two aircraft types, A and B, in its existing fleet (3 type A and 2 type B) and operates covering three routes as shown in Fig. 10. Table I shows the

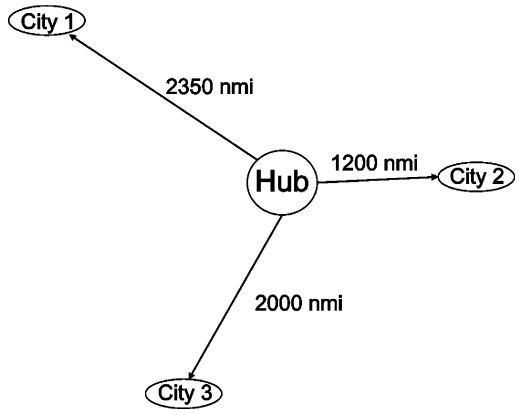


Fig. 10. Route configuration (nmi = nautical miles).

TABLE I
EXISTING AIRCRAFT A AND B KEY PARAMETERS

Aircraft parameters	Aircraft A	Aircraft B
Max. number of passengers	190	165
Aspect ratio	7.7	8
Wing loading	130	113
Thrust to weight ratio	0.31	0.30

parameters of the existing aircraft A and B. For simplicity, we assumed that the only design variable to be introduced for the aircraft is the maximum number of passengers (i.e., capacity).

A. All-in-One Problem

The all-in-one (AIO) problem aggregates aircraft allocation and design (i.e., selection) of the eight stages together. For each new aircraft type q introduced to the fleet in each stage s, maximum number of passengers y_q is defined such that it can only take the values of {160, 180, 200}, i.e., three choices of aircraft capacity (small, medium, large) at each stage. The objective of the AIO problem, DOC, is the summation of the daily DOC of all three routes of all existing aircraft and newly introduced aircraft throughout all eight stages. The new aircraft cost calculation is from [29].

A total of eight stages is considered in this case. In stage s, the maximum number of passengers (travel capacity) that can be transported for routes 1, 2, 3, by the existing fleet A and B, and the new q additional aircraft are given in (8)–(10). These travel handling capacities must satisfy the travel demand (given as fixed values for individual routes at each stage) that is allowed to change throughout the eight stages to reflect dynamic market demand. The number of routes here is $n = 3$ (Fig. 10)

$$Cap_1^s = y_A^s \times x_{A,1}^s + y_B^s \times x_{B,1}^s + \sum_{q=1}^{s} y_q^s \times x_{q,1}^s \quad (8)$$

$$Cap_2^s = y_A^s \times x_{A,2}^s + y_B^s \times x_{B,2}^s + \sum_{q=1}^{s} y_q^s \times x_{q,2}^s \quad (9)$$

$$Cap_3^s = y_A^s \times x_{A,3}^s + y_B^s \times x_{B,3}^s + \sum_{q=1}^{s} y_q^s \times x_{q,3}^s. \quad (10)$$

Here, the superscript indicates the stage, e.g., $x_{B,3}^2$ indicates the allocation variable for aircraft B on route 3 at stage 2. We labeled new aircraft as the first subscript index, thus at stage s, the newly introduced aircraft have indexes from 1 to q. These indexes will be used in the decomposed subproblems to denote individual stages. As seen from the AIO problem, the optimization problem becomes very large scale due to the combinatorial nature of the aircraft selection. Under the extreme exhaustive search, only for aircraft size variables y_q^s, the total number of possible combinations is $3^8 = 6561$, which confirms the need for the decomposed multistage approach presented in the next subsection

$$\begin{aligned}
P_{AIO}: \quad & \min \sum_{s=1}^{8} DOC_s \\
\text{w.r.t.} \quad & y_A^s, y_B^s, \quad s = 1, \ldots, 8 \\
& y_q^s, \quad s = q, \ldots, 8, \ \ q = 1, \ldots, 8 \\
& x_{A,i}^s, x_{B,i}^s, \quad s = q, \ldots, 8, \ \ i = 1, 2, 3 \\
& x_{q,i}^s, \quad s = q, \ldots, 8, \\
& q = 1, \ldots, 8, \ \ i = 1, 2, 3 \\
\text{s.t.} \quad & DOC_1(y_1^1) = \sum_{i=1}^{3} c_{A,i} \times x_{A,i}^1 + \sum_{i=1}^{3} c_{B,i} \times x_{B,i}^1 \\
& \qquad + \sum_{i=1}^{3} c_{1,i}(y_1^1) \times x_{1,i}^1, \ldots \\
& DOC_8(y_1^8, \ldots, y_8^8) = \sum_{i=1}^{3} c_{A,i} \times x_{A,i}^8 \\
& \qquad + \sum_{i=1}^{3} c_{B,i} \\
& \qquad \times x_{B,i}^8 + \sum_{i=1}^{3} c_{1,i}(y_1^8) \times x_{1,i}^8 + \cdots \\
& \qquad + \sum_{i=1}^{3} c_{8,i}(y_8^8) \times x_{8,i}^8 \\
& y_q^s \in \{160{,}180{,}200\}, \\
& s = q, \ldots, 8, \ \ q = 1, \ldots, 8 \\
& \sum_{i=1}^{3} x_{A,i}^s \le 6, \quad \sum_{i=1}^{3} x_{B,i}^s \le 4, \quad s = 1, \ldots, 8 \\
& \sum_{i=1}^{3} x_{1,i}^s \le 2, \ldots, \sum_{i=1}^{3} x_{8,i}^s \le 2, \quad s = 1, \ldots, 8 \\
& Cap_1^s \ge D_1^s, \quad s = 1, \ldots, 8 \\
& Cap_2^s \ge D_2^s, \quad s = 1, \ldots, 8 \\
& Cap_3^s \ge D_3^s, \quad s = 1, \ldots, 8 \\
& y_1^1 =, \ldots, = y_1^1, \quad y_2^2 =, \ldots, = y_2^8, \quad y_7^7 = y_7^8.
\end{aligned} \quad (11)$$

B. Multistage Decomposed Problem

The AIO problem is decomposed into stages, and each stage is optimized separately following the parallel coordination algorithm presented in the previous section. In the current example, the airline starts in stage one with two existing aircraft

types, A and B, and introduces new aircraft to the fleet. Up to the eighth stage, the airline keeps adding one new aircraft to the existing fleet. The decomposed problem for stage k is stated as follows:

$$
\begin{aligned}
P_s: \quad & \min DOC_s \\
\text{w.r.t.} \quad & y_A^s, y_B^s, \\
& y_q^s, \quad q = 1, \ldots, s \\
& x_{A,i}^s, x_{B,i}^s, \quad i = 1,2,3 \\
& x_{q,i}^s, \quad q = 1, \ldots, \ s, \ i = 1,2,3 \\
\text{s.t.} \quad & DOC_s(y_1^s, \ldots, y_q^s) \\
& = \sum_{i=1}^{3} c_{A,i} \times x_{A,i}^s + \sum_{i=1}^{3} c_{B,i} \times x_{B,i}^s \\
& \quad + \sum_{i=1}^{3} c_{1,i}(y_1^s) \times x_{1,i}^s + \cdots \\
& \quad + \sum_{i=1}^{3} c_{s,i}(y_q^s) \times x_{q,i}^s \\
& y_q^s \in \{160,180,200\}, \quad q = 1, \ldots, s \\
& \sum_{i=1}^{3} x_{A,i}^s \le 6, \quad \sum_{i=1}^{3} x_{B,i}^s \le 4, \\
& \sum_{i=1}^{3} x_{1,i}^s \le 2, \ldots, \sum_{i=1}^{3} x_{q,i}^s \le 2 \\
& Cap_1^s \ge D_1^s, \quad Cap_2^s \ge D_2^s, \quad Cap_3^s \ge D_3^s \\
& y_1^1 =, \cdots, = y_1^s, \quad y_2^2 =, \cdots, = y_2^s, \quad y_{q-1}^{q-1} = y_{q-1}^s.
\end{aligned}
$$

V. Numerical Results and Discussion

The airline allocation model with an eight stage optimization process has been used for the algorithm simulation purpose. Since the number of stages (tasks) is limited to eight, three different load balance mappings ($p = 2$, $p = 3$, $p = 4$) have been tested using the load balancing algorithm presented in the previous section. In general, it is difficult to predict the exact processing time of the task at each individual stage in this type of application. The linear increase of the processing time shown below in (12)–(13), although not exactly linear, provides a more reasonable expectation of better parallel performance with the load balance algorithm than random pair-up strategies for the parallelization.

Two demonstration problems with different demand scenarios were solved by the application and tested with the newly developed algorithm. The processing times at each stage of the problems are presented as

$$
\begin{aligned}
ProblemA: \quad & t(T_A) = \{t(T_1), t(T_2), \ldots, t(T_8)\} \\
& = \{234,246,317,654,1150, \\
& \quad 2598,2470,6734\} \text{ [sec.]} \qquad (12) \\
ProblemB: \quad & t(T_B) = \{t(T_1), t(T_2), \ldots, t(T_8)\} \\
& = \{48,83,268,531,1042, \\
& \quad 1911,4892,3906\} \text{ [sec.]}. \qquad (13)
\end{aligned}
$$

TABLE II
Results of the Speedups for the Tested Cases

	Problem A	Problem B
Sp(1)	1.0	1.0
Sp(2)	1.46	1.83
Sp(3)	1.98	1.99
Sp(4)	2.07	2.4

The cases of $p = 2$ and $p = 4$ fall into the category that utilizes the mirror image mapping (even m, even p). For $p = 3$, since the condition $\text{mod}(n,p) = 0$ is not satisfied, zero-load task padding is required. Adding only one additional pseudotask processing results in a case of $m = 3$ and $p = 3$, which falls into Case II of the algorithm.

The speedup, $S_p(p)$, of the each case has been analyzed for the parallel performance test. The speedup is the ratio of processing time for sequential execution to processing time for parallelized execution of the application using p processors [30]. Table II summarizes the results of the two test cases under the following speedup definition:

$$
Sp(p) = \frac{\sum_{i \in t(T)} i}{\max_{1 \le j \le p} \sum_{i \in ST} i} \qquad (14)
$$

where the largest value of the processing time in the subtask group was selected in the denominator.

As shown in Table II, the speedups of the two test problems increase as the number of processors increases. Problem B displays better speedups because the processing time of the stages in the problem is closer to the strict linear case than in problem A.

The size of the application limited the number of test cases in this study, but the simulation with the two test cases shows that the load balancing algorithm ensures a good task-parallelization for the multi-stage optimization applications with linear increase of processing time.

VI. Conclusion

The proposed quasi-separable multistage framework simultaneously enables finding an optimal enterprise design and planning. Also, the task-parallel algorithm allows an optimal load balancing to solve the multistage programming problem in a parallel, decomposed manner to overcome the challenge of increased complexity of the solution process. The airline demonstration case showed that the proposed framework and solution algorithm finds an optimal system selection (or design) and its optimal allocation in conjunction with the existing fleet. For the eight-stage problem, it was shown that the task-parallel algorithm achieved a good speedup with an increasing number of processors. The proposed methodology is significant in that it allowed a joint decision making for future operating scenarios for previously separate design and planning problems. Future research involves acquisition and operation problems for the new system of systems, and parallel optimization algorithm development for an efficient solution process. Also, an interesting follow-up would be to identify the correlation between system

of systems capability to predict emerging behavior and operational conditions such as future passenger demand, which would enable even longer term planning and design.

REFERENCES

[1] W. Crossley, M. Mane, and Nusawardhana, "Variable resource allocation using multidisciplinary optimization: Initial investigations for system of systems," in *Proc. A 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conf.*, 2004, pp. 1–17.

[2] I. J. Hidalgo and H. M. Kim, "System of systems model for vehicle design with resource allocation," in *Proc. 2006 ASME Design Automation Conf.*, Philadelphia, PA, Sep. 2006.

[3] H. M. Kim and I. J. Hidalgo, "System of systems engineering model by multistage analytical target cascading," in *Proc. INCOSE 2007*, San Diego, CA, Jun. 2007.

[4] H. M. Kim, N. F. Michelena, P. Y. Papalambros, and T. Jiang, "Target cascading in optimal system design," *Trans. ASME: J. Mech. Des.*, vol. 125, no. 3, pp. 474–480, 2003.

[5] M. Lawson and H. Karandikar, "A survey of concurrent engineering," *Concurr. Eng.*, vol. 2, no. 1, pp. 1–6, 1994.

[6] F. Elgh, "Concurrent cost estimation as a tool for enhanced producibility-system development and applicability for producibility studies," *Int. J. Prod. Econ.*, vol. 10, no. 1–2, pp. 12–26, Sep. 2007.

[7] S. K. Fixson, "Modularity and commonality research: Past developments and future opportunities," *Concurr. Eng.: Res. Applicat.*, vol. 15, no. 2, pp. 85–85, 2007.

[8] D. Xue and H. Yang, "A concurrent engineering-oriented design database representation model," *CAD Comput. Aided Des.*, vol. 36, no. 10, pp. 947–965, 2004.

[9] F. B. Manolache and S. Costiner, Parallel Processing Approaches for Multi Disciplinary Optimization Algorithms 2001 [Online]. Available: http://www.math.cmu.edu

[10] H. Parsaei and W. Sullivan, *Concurrent Engineering*. New York: Chapman & Hall, 1993.

[11] R. Schmidt and M. Schmidt, *Computer Aided Concurrent Integral Design*. Berlin, Germany: Springer-Verlag, 1996.

[12] Y. Nahm, "Integrated product and process modeling for collaborative design environment," *Concurr. Eng.: Res. Applicat.*, vol. 12, no. 11, pp. 5–23, Mar. 2004.

[13] P. K. Umesha, M. T. Venuraju, D. Hartmann, and K. R. Leimbach, "Parallel computing techniques for sensitivity analysis in optimum structural design," *J. Comput. Civil Eng.*, no. 6, pp. 463–477, 2007.

[14] J. J. E. Dennis and Z. Wu, *Parallel Continuous Optimization*. San Mateo, CA: Morgan Kaufmann, 2003, pp. 649–670.

[15] M. D'Apuzzo and M. Marino, "Parallel computational issues of an interior point method for solving large bound-constrained quadratic programming problems," *Parallel Comput.*, vol. 29, no. 4, pp. 467–483, 2003.

[16] V.-D. Cung, S. Martins, C. Ribeiro, and C. Roucairol, *Strategies for the Parallel Implemetation of Metaheuristics*, ser. Essays and Surveys in Metaheuristics. Norwell, MA: Kluwer, 2002, pp. 263–308.

[17] T. Crainic, M. Gendreau, and J.-Y. Potvin, *Parallel Tabu Search*, ser. Parallel Metaheuristics. New York: Wiley, 2005.

[18] T. Ralphs, L. Ladanyi, and M. Salzman, "Parallel branch, cut, and price for large-scale discrete optimization," *Mathemat. Programm.*, no. B98, pp. 253–280, 2003.

[19] J. Sobieszczanski-Sobieski, J. Agte, and R. Sandusky, Jr, "Bilevel integrated system synthesis," *AIAA J.*, vol. 38, no. 1, pp. 164–172, 2000.

[20] J. Sobieszczanski-Sobieski, "Optimization by decomposition: A step from hierarchic to non-hierarchic systems," in *Proc. 2nd NASA Air Force Symp. Recent Advances in Multidisciplinary Analysis and Optimization*, Hampton, VA, 1988.

[21] R. Braun, "Collaborative Optimization: An Architecture for Large-Scale Distributed Design," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1996.

[22] H. Kim, "Target Cascading in Optimal System Design," Ph.D. dissertation, Univ. Michigan, Ann Arbor, 2001.

[23] A. de Wit and F. van Keulen, "Numerical comparison of multi-level optimization techniques," in *Proc. 48th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conf.*, Honolulu, HI, Apr. 23–26, 2007.

[24] H. Kim, B. Malone, and J. Sobieszczanski-Sobieski, "A distributed, parallel, and collaborative environment for design of complex systems," in *Proc. 45th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conf.*, Palm Springs, CA, Apr. 2004.

[25] S. Jin, K. Kim, K. Jeong, J. Lee, J. Kim, H. Hwang, and H. Suh, *MEDIC: A MDO-Enabling Distributed Computing Framework*, ser. Fuzzy Systems and Knowledge Discovery. Berlin/Heidelberg, Germany: Springer, 2005, pp. 1092–1101.

[26] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems," in *Proc. Heterogeneous Computing Workshop*, 1999, pp. 15–29.

[27] H. D. Kim and J. S. Kim, "An online scheduling algorithm for grid computing systems," in *Proc. Int. Conf. Grid and Cooperative Computing*, 2003, pp. 34–39.

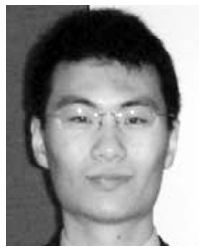
[28] B. Hamidzadeh, L. Y. Kit, and D. J. Lilja, "Dynamic task scheduling using online optimization," *IEEE Trans. Parallel Distrib. Syst.*, vol. 11, no. 11, pp. 1151–1163, Nov. 2000.

[29] D. P. Raymer, *Aircraft Design: A Conceptual Approach*. Houston, TX: AIAA, 1999.

[30] G. K. A. Grama, A. Gupta, and V. Kumar, *Introduction to Parallel Computing*. Reading, MA: Addison-Wesley, 2003.

Harrison M. Kim received the Ph.D. degree in engineering system design and optimization in mechanical engineering from the University of Michigan, Ann Arbor, in 2001.

He joined the University of Illinois in 2005 and has been leading the Enterprise Systems Optimization Lab. Previously, he held positions in research and consulting at the U.S. Army Automotive Research Center, Northwestern University, and a business-IT consulting company. His research interests include a variety of areas of system design and optimization utilizing mathematical programming, simulation, data mining, and informatics.



Shen Lu received the B.E. degree in automation and the M.E. degree in control science and engineering from Tsinghua University, Beijing, China, in 2003 and 2006, respectively. He is currently pursuing the Ph.D. degree in industrial engineering at the University of Illinois at Urbana-Champaign.

His research interests include product design optimization, mathematical programs with complementarity constraints, and energy systems.

Jin Suk Kim received the Ph.D. degree in computer science from Korea Advanced Institute of Science and Technology (KAIST), Seoul, in 1990.

He is currently a Professor in the School of Computer Science, University of Seoul, where he is also serving as the Deputy Vice President for Research. His research spans broad areas of computer science including high performance computing, wireless sensor network, and computer algorithm.

Byoung-Do Kim received the Ph.D. degree in aeronautics and astronautics engineering from Purdue University, West Lafayette, IN, in 2002.

He joined the National Center for Supercomputing Applications (NCSA), University of Illinois at Urbana-Champaign, in 2003 as a Research Programmer. His research focus includes HPC applications performance engineering, application optimization and scalability, and grid computing technologies. He joined Texas Advanced Computing Center (TACC), The University of Texas at Austin, in 2007, where his current research interest is applications performance optimization on multicore systems. His major field of research is aerospace systems and computational fluid dynamics in large-scale high-performance computing environments.