Parallel Multidisciplinary Design Optimization

Weizhen Mao *and* David M. Nicol
Department of Computer Science
The College of William and Mary
Williamsburg, Virginia 23187-8795
{*wm, nicol*}*@cs.wm.edu*

Abstract

In this paper, we define a parallel version of the multidisciplinary design optimization problem, where parallel processors are available to execute computation modules with input/output communications in a complex engineering system. In addition, we study the computational complexity issue and present a few heuristics for the problem.

Keywords: Mapping and scheduling, resource allocation, optimization, algorithms.

1 Introduction

Today's complex engineering systems often consist of hundreds or even thousands of subsystems (computation modules) coupled with links that transfer subsystem output data. This new area, called the *Multidisciplinary Design Optimization* (MDO) [5], allows the entire engineering system to be optimized globally as one unit even though the analysis associated with each subsystem may be performed locally using separate hardware and software.

A very important feature of an engineering system is called *convergence*. Consider the following example. In building an airplane, different groups of people are interested in different features of the airplane. For example, Fluid Dynamic people care about the shape of the plane, while the Mechanical Engineering people are interested in the structure. Since one satisfying design from one group may not be a good design to the other, the two groups have to work together and the design goes back and forth until both ends are happy. This process is called *convergence*. In Figure 1 (a), node 1 and node 2 are computation modules, representing the two disciplines, Fluid Dynamic and Mechanical Engineering, respectively. The output of module 1 serves as the input to module 2, and the output of module 2 serves as the input to module 1. The cycle between nodes 1 and 2 loops an unknown number of times before reaching convergence. The graph in Figure 1 (b) is called the *Design Structure Matrix* (DSM). It represents the same system as in Figure 1 (a). In a DSM, nodes, representing modules, are placed on the diagonal of an imaginary matrix, and dots, representing arcs, connect associated nodes. For instance, in Figure 1 (b), the dot above the diagonal, also called a *feedforward*, is the arc from node 1 to node 2, and the dot below the diagonal, also called a *feedback*, is the arc from node 2 to node 1.

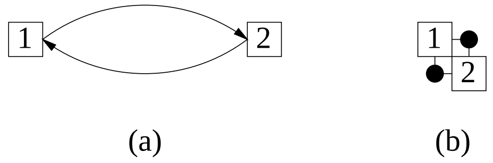

Figure 1: (a) A cycle of two nodes. (b) The DSM for the cycle of two nodes.

The past research has mainly focused on making the system as simple as possible by reducing the number of feedbacks [4] and reducing the number of couplings [1] in the DSM of the system. As for how the computation modules can be executed in the most efficient way, previous research only assumes sequential execution, i.e., the modules are executed in the order they appear on the diagonal of the DSM, and once a feedback link is encountered the execution falls into a loop until the corresponding cycle converges. When there are multiple feedback links in several nested cycles, the execution flow becomes rather complicated since loops may overlap and there is no way to know beforehand when a loop reaches convergence.

2 Parallel Execution of a DSM

While the previous research assumes that there is only one processor (machine) available to execute the modules in a complex engineering system, we consider the parallel processing environment, where multiple processors are available to execute the modules. We wish first to define a partition of the module set and map each subset to a processor, and second to construct a time schedule for the modules mapped to each processor. Our goal is to make the total elapsed time from the start of the system to the convergence of the system, or the makespan of the schedule, as small as possible. We call this problem the parallel MDO.

Consider a graph that represents an engineering system. The weight of a node is clearly the computation time of the corresponding module. Each directed edge (link) has two weights: the internal communication cost which incurs when the two nodes the edge connects are later assigned to the same processor, and the external communication cost which incurs when the two nodes the edge connects are assigned to different processors. Obviously, the former is smaller than the latter. When multiple processors are involved, it makes sense to assign nodes connected by links with high external communication costs to the same processor and nodes in a cycle to the same processor. On the other hand, when two nodes are not connected by a path, they can be executed in parallel on different processors. In some architectures such as a network of workstations, communication may have to be monitored by a processor, while some parallel machines allow the modules to handle communication on their own. Further, modules may have multiple communication ports so that several messages may be sent and received simultaneously.

This problem is very similar to the multiple machine scheduling of jobs with precedence constraints [3] except that the graph in the machine scheduling problem describes the precedence constraints among jobs and thus must be acyclic, while the graph in the parallel MDO reflects the input/output dependency and thus allows the presence of cycles. For example, in Figure 1, node 1's output serves as node 2's input and node 2's output is node 1's input. Logically speaking, the computation in the cycle can never get started since the computation in node 2 relies on the computation in node 1 and vice versa. However, we can overcome this by assuming that initially there are some default data on the edges to enforce a successful start of the computation and once the computation begins the data dependency among modules in a cycle cannot be ignored.

3 Complexity Results

In this section, we will restrict our attention to the decision problem, i.e., given a bound B, is there a solution to the parallel MDO with the makespan no larger than B? Further, we assume that the communication is handled by each module independently. That is, each communication link can be viewed as a conveyer belt which moves at a constant speed. Whenever the module completes its computation, it puts its output on the belt, which will deliver the data to the module that needs it as input, without the attention of the processor and the module. We show that even the following simplest versions of the parallel MDO are NP-complete.

1. Modules in the system are all independent, i.e., $E = \emptyset$ in the graph.
2. The graph is acyclic and $E \neq \emptyset$.
3. The graph has cycles, but may not be connected otherwise.
4. The graph has cycles and is connected.

Each of the above NP-completeness results can be obtained by establishing polynomial reduction from the well-known *Multiprocessor Scheduling* problem [2] defined as follows.

Multiprocessor Scheduling

INSTANCE: Set J of n jobs, processing time $t(j) \in Z^+$ for each $j \in J$, m processors, and a deadline $D \in Z^+$.

QUESTION: Is there an m-processor schedule for J that meets the overall deadline D, i.e., a function $s : J \rightarrow Z_0^+$ such that, for all $u \geq 0$, the number of jobs $j \in J$ for which $s(j) \leq u < s(j)+t(j)$ is no more than m and such that, for all $j \in J$, $s(j) + t(j) \leq D$?

To save space, we will not follow the standard NP-completeness proving procedure, but instead only describe the polynomial reduction, which is usually the key in an NP-complete proof.

For the first case when $E = \emptyset$, the reduction is almost straightforward. Let the graph contain just n nodes without edges, each corresponding to a job. Let the computation time of each node be the processing time of the corresponding job. Finally, let $p = m$ (p is the number of processors in the parallel MDO) and $B = D$. The proof that there is an m-processor schedule that meets the deadline D if and only if there is a partition/schedule for the graph G with makespan no greater than B is omitted. The key in the proof is that each subset of jobs assigned to one processor corresponds to a subset in the partition of G. The schedule of the modules within each subset is simply any sequential execution of the modules since the modules are not connected by any edges.

The reduction for the second case when the graph is acyclic and $E \neq \emptyset$ is similar to the first case above, except that one more node v is added to the graph. Also add edges from each of the n original nodes to v. Let the internal and external communication times on each edge both be $\epsilon \leq \min_{j \in J}\{t(j)\}$. Let the computation time of v be 1. Finally, let $p = m$ and $B = D + \epsilon + 1$. The if and only if proof is also similar to that in the previous proof. Each subset of jobs executed on one processor in the Multiprocessor Scheduling defines a subset of modules in the partition for the parallel MDO. The additional module v is put in any subset. When all the n original nodes have completed their computation, ϵ time units are needed for the node that is completed last to transmit its output to v. This is why B is set to be $D + \epsilon + 1$.

In the third case, we assume that the graph has cycles, but may not be connected otherwise. We define a graph with $n + 1$ nodes, one more than the number of jobs. The first $n - 1$ nodes corresponds to the first $n - 1$ jobs with the respective computation costs, and the last two nodes correspond to the last job. These two nodes are connected in a cycle by two edges. The internal communication cost on each edge is ϵ, a small positive number, and the external communication cost on each edge is η, a very large number, which prevents the two nodes from being assigned to two different subsets in the partition. The computation costs of the two nodes are $\frac{1}{4}t_n - \frac{3}{2}\epsilon$ and $\frac{1}{4}t_n$, where t_n is the processing time of the last job J_n. Also, assume that the cycle loops just once to reach convergence. Clearly, the total time needed to complete the computation and communication in the cycle is $2(\frac{1}{4}t_n - \frac{3}{2}\epsilon + \frac{1}{4}t_n) + 3\epsilon = t_n$, assuming the two nodes are assigned to the same processor. Finally, let $p = m$ and $B = D$. The proof that there is an m-processor schedule that meets the deadline D if and only if there is a partition/schedule for the instance defined above with makespan no greater than B is straightforward and is therefore omitted.

For the final case when the graph has cycles and is connected, we take the graph constructed for the third case and add a new node v together with edges from each of the first $n - 1$ nodes and one of the last two nodes to the new node v. Let the computation cost of v be 1 and the internal and external communication costs on all newly added edges be ϵ. Finally, let $p = m$ and $B = D + \epsilon + 1$. The proof that there is an m-processor schedule that meets the deadline D if and only if there is a partition/schedule for the instance defined above with makespan no greater than B is straightforward and is therefore omitted.

4 Heuristics

Due to the complexity difficulty of the parallel MDO, we focus our attention on designing heuristics. One heuristic is to recognize cycles in the graph and assign all nodes in one cycle to a processor. Another heuristic is to estimate the number of iteration for each cycle and "unroll" the cycles into an acyclic graph so that heuristics for the similar machine scheduling problem may be applied. Because the convergence feature of the system adds great difficulty to the problem, analyzing a heuristic mathematically is almost impossible. As a direction for future research, we will adopt simulation methods to evaluate the performance of a heuristic in practice.

Another approach we take is to study special cases of the problem. For example, the communication costs of the links may be neglected since they are often very small in comparison with the computation costs. Also, we may consider the case that cycles in the graph do not overlap, i.e., there is no node that appears in more than one cycle. In this case, we can convert the parallel MDO to the classical machine scheduling with acyclic precedence constraints by contracting cycles into condensed nodes and modifying weights of the new nodes.

5 Conclusions

Previous research on multidisciplinary design optimization has focused on attempts to simplify the engineering system by reducing the number of input/output links, thus the number of loops. Once this preprocessing stage is completed, computation modules in the system are then being executed sequentially by a single processor. In our research, we pay attention to the execution stage of the system. To achieve high efficiency, we allow the modules to be executed by a set of parallel processors. In this paper, we defined the so-called parallel MDO, studied the computational complexity issue, and presented a few heuristics for the problem.

References

[1] C. L. Bloebaum, An intelligent decomposition approach for coupled engineering systems, The 4th AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization, Cleveland, OH, 1992.

[2] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, 1979.

[3] E. L. Lawler, J. K. Lenstra. A. H. G. Rinnooy Kan and D. B. Shmoys, Sequencing and scheduling: algorithms and complexity, in *Handbooks in Operations Research and Management Science, Volume 4: Logistics of Production and Inventory*, S. C. Graves, A. H. G. Rinnooy Kan and P. Zipkin, ed., North-Holland, 1990.

[4] C. M. McCulley, A genetic tool for optimally sequencing the design of complex engineering systems, Master thesis, The State University of New York at Buffalo, 1995.

[5] R. H. Tolson and J. Sobieski, Multidisciplinary analysis and synthesis: Needs and opportunities, The 26th AIAA/ASME/ASCE/AHS Structures, Structural Dynamics and Material Conference, Orlando, FL, 1985.