Deployment Optimization for Embedded Flight Avionics Systems

[1]Brian Dougherty, [2]Jules White, [1]Douglas C. Schmidt, [3]Russell Kegley and [3]Jonathan Preston
[1]Vanderbilt University, {briand,schmidt}@dre.vanderbilt.edu
[2]Virginia Tech, julesw@vt.edu
[3]Lockheed Martin Aeronautics, {russell.b.kegley,jonathan.d.preston}@lmco.com

1 Abstract

Loosely-coupled publish/subscribe messaging systems facilitate optimized deployment of software applications to hardware processors. Intelligent algorithms can be used to refine system deployments to reduce system cost and resource requirements, such as memory and processor utilization. This paper presents the optimization of a legacy flight avionics system deployment with the Scatter Deployment Algorithm(ScatterD), resulting in a reduction of required processors and network bandwidth consumption.

2 Introduction

Current trends and challenges. Several trends are shaping the development of embedded flight avionics systems. First, there is a migration away from older *federated computing architectures* where each subsystem occupied a physically separate hardware component to *integrated computing architectures* where multiple software applications implementing different capabilities share a common set of computing platforms. Second, publish/subscribe (pub/sub)-based messaging systems are increasingly replacing the use of hard-coded cyclic executives.

These trends are yielding a number of benefits. For example, integrated computing architectures create an opportunity for system-wide optimization of *deployment topologies*, which map software components and their associated tasks to hardware processors as shown in Figure 1. Optimized deployment topologies can pack more software components onto the hardware, thereby optimizing system processor, memory, and I/O utilization. Increasing hardware utilization can decrease the total hardware processors that are needed, lowering both implementation costs and maintenance complexity. Moreover, reducing the required hardware infrastructure has other positive side effects, such as weight and power consumption savings. Decoupling software from specific hardware processors also increases flexibility by not coupling embedded software application components with specific hardware processing platforms.

Open problems. The explosion in the size of the search space for large-scale embedded deployment topologies makes it hard to optimize them without computer-

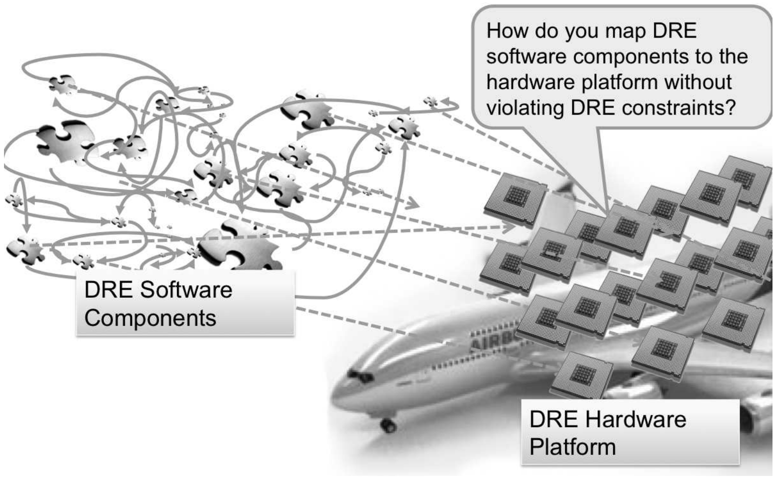
How do you map DRE software components to the hardware platform without violating DRE constraints?
DRE Software Components
DRE Hardware Platform

Figure 1. Flight Avionics Deployment Topology

assisted methods to evaluate the schedulability, network bandwidth consumption, and other characteristics of a given configuration. Developing computer-assisted methods to deploy software to hardware in embedded systems is a challenging problem [1, 4] due to the large number of complex constraints that must be addressed.

For example, developers must ensure that each software component is provided with sufficient processing time to meet any real-time scheduling constraints [7]. Likewise, resource constraints (such as total available memory on each processor) must also be respected when mapping software components to hardware components [7, 5]. Moreover, assigning real-time tasks in multiprocessor and/or single-processor machines is NP-Hard [3], which means that such a large number of potential deployments exist that it would take many years to investigate all possible solutions.

Current algorithmic deployment techniques are largely based on bin-packing [3, 6, 2], which represents the software tasks as items that take up a set amount of space and hardware processors as bins that provide limited space. All of the items are then packed into as few bins as possible while maintaining that the sum of the space consumed by the items in a bin does not exceed the space provided by the bin in which they are placed. Bin-packing deployment techniques take a one-dimensional view of deployment problems by focusing on a single deployment concern at a time, such as resource constraints, scheduling constraints, or fault-tolerance constraints. In production flight avionics systems, however, deployments must meet a combination of these concerns.

Solution approach ⇒ Deployment Optimization with ScatterD. This paper describes and validates the Scatter Deployment Algorithm (ScatterD), a tool we developed to perform computer-assisted deployment optimization for flight avionics systems. This tool combines heuristic bin-packing with optimization algorithms, such as genetic algorithms (GAs) [**?**]. We show how developers of flight avionics systems can use optimization tools like ScatterD to reduce the processor and network bandwidth requirements of deployments.

The remainder of this paper is organized as follows: Section 3 outlines a flight avionics deployment case study we use to motivate the challenges and solutions throughout the paper; Section 4 describes the challenges faced by developers when attempting to optimize a flight avionics deployment topology; Section 5 discusses the ScatterD tool for deployment optimization; Section 6 provides empirical results demonstrating the reductions in hardware footprint and network bandwidth consumption that our ScatterD can produce; and Section 7 presents concluding remarks.

3 Modern Embedded Flight Avionics Systems: A Case Study

Over the past 20 years, flight avionics systems have become increasingly sophisticated, to the point where many modern aircraft depend heavily on software executing atop a complex embedded network for higher-level capabilities, such as more sophisticated flight control and more advanced weapon systems. The increased weight of the embedded computing platforms required by a modern fighter aircraft incurs a multiplier effect, *e.g.*, roughly four pounds of cooling, power supply, and other supporting hardware are needed for each pound of processing hardware, reducing mission range, increasing fuel consumption, and impacting aircraft responsiveness. To accommodate the increased amount of software required, avionics systems have moved from older federated computing architectures to integrated computing architectures that combine multiple software applications together on a single computing platform containing many software components.

The class of flight avionics system targeted by our work is a networked parallel message-passing architecture containing many computing nodes, as shown in Figure 2.

Each node is built from commercially available components packaged in hardened chassis to withstand extremes of temperature, vibration, and acceleration. At the individual node level, ARINC 653-compliant time and space partitioning separates the software applications into sets with compatible safety and security requirements. Inside a given time partition, the applications run within a hard real-time deadline scheduler that executes the applications at a variety of harmonic periods.

The integrated computing architecture shown in Figure 2 has benefits and challenges. Key benefits include better optimization of hardware resources and increased flexibility, which result in a smaller hardware footprint, lower energy use, decreased weight, and enhanced ability to add new software to the aircraft without updating the hardware. The key challenge, however, is increased system integration complexity. In particular, while the homogeneity of processors gives system designers a great deal of freedom allocating software applications to computing nodes, optimizing this allocation involves simultaneously balancing multiple competing resource demands.

For example, even if the processor demands of a pair of applications would allow them to share a platform, their respective I/O loads may be such that worst-case arrival rates would saturate the network bandwidth flowing into a single node. This problem is complicated for single-core processors used in current integrated computing architectures. Moreover, this problem is becoming more complicated with the adoption and fielding of multi-core processors, where competition for shared resources expands to include internal buses, cache memory contents, and memory access bandwidth.

4 Deployment Optimization Challenges

While Section 3 describes many of the benefits that can be acquired through deployment optimization, developers of embedded flight avionics systems face a daunting series of conflicting constraints and optimization goals when determining how to deploy software to hardware that make deployment optimization difficult. It is hard to find a valid solution for a single deployment constraint, such as ensuring that all of software tasks can be scheduled to meet real-time deadlines, in isolation using conventional techniques, such as bin-packing. It is much harder, however, to find a valid solution when considering many deployment constraints, such as satisfying resource requirements of software tasks in addition to ensure schedulability. Moreover, optimizing the deployment topology of a system to minimize consumed network bandwidth or other dynamic properties is more difficult since communication between software tasks must also be taken into account instead of simply considering each software task as an independent entity. This section describes the challenges that developers face when attempting to derive a deployment topology for a flight avionics

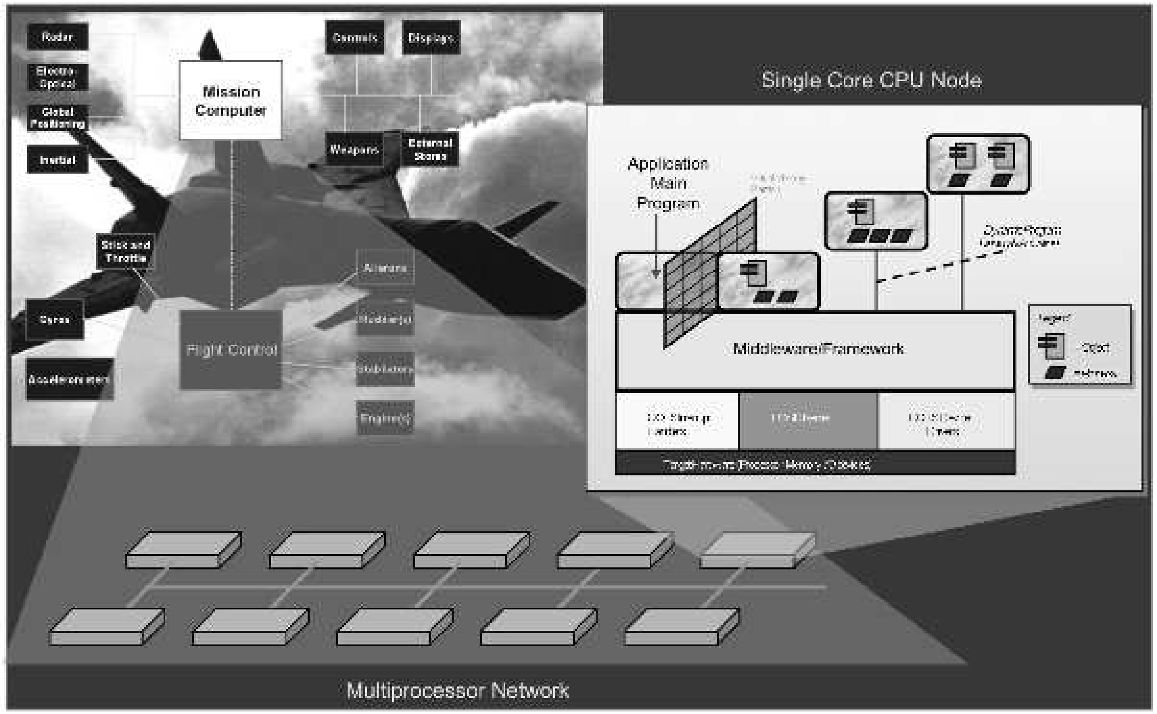

Figure 2. An Integrated Computing Architecture for Embedded Flight Avionics

system with a networked parallel message-passing architecture as described in Section 3 that minimizes the number of required processors and the total network bandwidth resulting from communication between software tasks.

4.1 Challenge 1: Rate-monotonic Scheduling Constraints

In real-time systems, such as the embedded flight avionics case study from Section 3, either fixed priority scheduling algorithms, such as rate-monotonic (RM) scheduling, or dynamic priority scheduling algorithms, such as earliest-deadline-first (EDF), control the execution ordering of individual tasks on the processors. The deployment topology must ensure that the set of software components allocated to each processor are schedulable and will not miss real-time deadlines. Finding a deployment topology for a series of software components that ensures schedulability of all tasks is called “multiprocessor scheduling” and is NP-Hard [3].

A variety of algorithms, such as bin-packing algorithm variations, have been created to solve the multiprocessor scheduling problem. A key limitation of applying these algorithms to deployment optimization problems is that bin-packing does not allow developer to specify the characteristic of the deployment to be optimized. For example, bin-packing does not allow developers to specify an objective function based on the overall network bandwidth consumed by a deployment topology to determine a deployment topology that minimized bandwidth. We describe how ScatterD guarantees schedulability in Section 5.1.

4.2 Challenge 2: Memory, Cost, and Other Resource Constraints

Scheduling a processor is not the only type of resource that must be managed while searching for a deployment topology. Hardware nodes often have other limited but critical resources, such as main memory or core cache, necessary for the set of software components it supports to function. Developers must ensure that the components deployed to a processor do not consume more resources than are present. If each processor does not provide a sufficient amount of these resources to support all tasks on the processor, a task will not execute properly, resulting in a failure. Moreover, since each processor used by a deployment has a cost associated with it, developers may need to adhere to a global budget, as well as scheduling constraints. We describe how ScatterD ensures that resources constraints are satisfied in Section 5.1.

4.3 Challenge 3: Network Resource and Topology Constraints

Embedded flight avionics systems often must ensure that not only processor resource limitations are adhered to but network resources, such as bandwidth are not over-consumed. For example, if two critical real-time components communicating across a high-speed bus, such as a controller area network (CAN) bus, fail to send a required message due to network saturation, catastrophic failure could occur. The consumption of network resources is determined by the number of interconnected components that are not colocated on the same processor. For example, if two components are colocated on the same processor, they do not consume any bandwidth.

Adding the consideration of network resources to deployment substantially increases the complexity of finding a software-to-hardware deployment topology mapping that meets requirements. With real-time scheduling and resource constraints, the deployment of a component to a processor has a fixed resource consumption price that can be calculated in isolation of the other components. The impact of the component's deployment on the network, however, cannot be calculated in isolation of the other components. The impact is determined by finding all other components that it communicates with, determining if they are colocated, and then calculating the bandwidth consumed by the interactions with those that are not colocated. We descibe how ScatterD can be used to minimize the bandwidth required by a system deployment in Section 5.2.

5 Deployment Optimization for Bandwidth and Processor Minimization

Heuristic bin-packing algorithms work well for multiprocessor scheduling and resource allocation. As discussed in Section 4, however, heuristic bin-packing is not effective for optimizing designs for certain system-wide properties, such as network bandwidth consumption, and hardware/software cost. Metaheuristic algorithms, such as genetic algorithms or particle swarm optimization techniques, allow designers to optimize these system-wide properties that are not easy to optimize with bin-packing algorithms [**?**, **?**].

To overcome the limitations of applying either a heuristic algorithm, such as bin-packing, or a metaheuristic algorithm, such as a genetic algorithm, individually to deployment optimization, this section presents ScatterD, a tool that utilizes a "hybrid" method that combines the two approaches so the benefits of each can be obtained with a single algorithm. We explain how we developed the ScatterD tool that integrates the heuristic bin-packing algorithm's ability to generate correct solutions to scheduling and resource constraints with the metaheuristic algorithm's flexible optimization capabilities for minimizing network bandwidth and processor reduction.

5.1 Satisfying Deployment Constraints with ScatterD

To optimize the deployment of the flight avionics system described in Section 3 we developed ScatterD. ScatterD ensures that the numerous deployment constraints, such as schedulability and resource constraints described in Section **??** are satisfied by using heuristic bin-packing to allocate software tasks to processors. Existing bin-packing algorithms for multiprocessor scheduling are designed to take as input a series of items (*e.g.*, tasks or software components), the set resources consumed by each item (*e.g.*, processor and memory), and the set of bins (*e.g.*, processors) and their capacities. The algorithm outputs an assignment of items to bins (*e.g.*, a mapping of software components to processors).

With ScatterD, we ensure that schedulability is guaranteed by using response-time analysis to ensure a software component can be scheduled on a given processor before allocating its associated item to a bin. Before placing an item in a bin, ScatterD analyzes the response time that would result from allocating the software task to the given proessor. If the response time is fast enough to meet the real-time deadlines of the software task then the software task can be allocated to the processor. If not, then the item must be placed in another bin.

To ensure that the resource constraints, such as memory requirements, of each software task is met, we specify a capacity for each bin that is defined by the amount of each computational resource provided by the processor. Similarly, the resource demands of each software task define the resource consumption of each item. Before an item can be placed in a bin, ScatterD verifies that the total consumption of each resource utilized by the item to be placed and any items already placed in the bin does not exceed the resources provided.

5.2 Network Bandwidth and Processor Minimization with ScatterD

As discussed in Section 5.1, ScatterD uses heuristic bin-packing to ensure that schedulability and resource constraints are met. Bin-packing, however, will always yield the same solution for a given set of software tasks and processors if the heuristics is not altered. Therefore, the number of processors utilized and the network bandwidth requirements will not change from one execution of the bin-packing algorithm to another. In the vast deployment solution space, however, there may be many other deployments that still satisfy all design constraints while substantially reducing the number of processors and network bandwidth required.

Metaheuristic algorithms, such as genetic algorithms and particle swarm optimization techniques, can be used to explore other areas of the deployment solution space and discover solutions that require less processors and network bandwidth to function. The problem, however, is that that deployment space is incredibly vast containing only a small percentage of potential deployments that would satisfy all design constraints. Since metaheuristic algorithms strive to reduce bandwidth and the number of required processors without accounting for design constraints, using metaheuristic algorithms alone would result in many invalid deployments.

To allow metaheuristic algorithms to search the deployment space for deployments with minimal processor and bandwidth requirements while still ensuring that design constraints are met, ScatterD uses metaheuristic algorithms to *seed* the bin-packing algorithm. Metaheuristic algorithms are used to search the deployment space and select several software tasks that must be packed prior to the rest of the software tasks. By forcing an altered bin-packing order, new deployments with different bandwidth and processor requirements are generated. Since bin-packing is still the driving force behind allocating software tasks, design constraints have a higher probability of being satisfied.

As new valid deployments are discovered, they are scored based on network bandwidth consumption and the number of required processors. Metaheuristic algorithms use the scores of these deployments to determine which new packing order would likely lead to more a optimized deployment. By using metaheuristic algorithms to search the design space and then using bin-packing to allocate software tasks to processors, ScatterD can generate deployments that meet all design constraints while also minimizing network bandwidth consumption and reducing the number of required processors.

6 Empirical Results

This section presents the results of combining two metaheuristics algorithms (particle swarm optimization and a genetic algorithm) with bin-packing to create two different versions of the ScatterD tool to optimize the deployment of the embedded flight avionics system described in Section 3. We applied these techniques to determine if (1) a deployment exists that increases processor utilization to the extent that legacy processors could be removed and (2) the overall network bandwidth requirements of the deployment were reduced due to colocating communicating software tasks on a common processor.

The first experiment examined applying ScatterD to minimize the number of processors in the fligt avionics system deployment. The legacy flight avionics deployment consisted of software tasks deployed to 14 processors. Applying ScatterD with particle swarm optimization techniques and genetic algorithms resulted in increased utilization of the processors, allowing the software to be deployed to only eight processors in both cases. The remaining six processors could then be removed from the deployment without effecting system performance, resulting in a 42.8% reduction as shown in Figure 3.

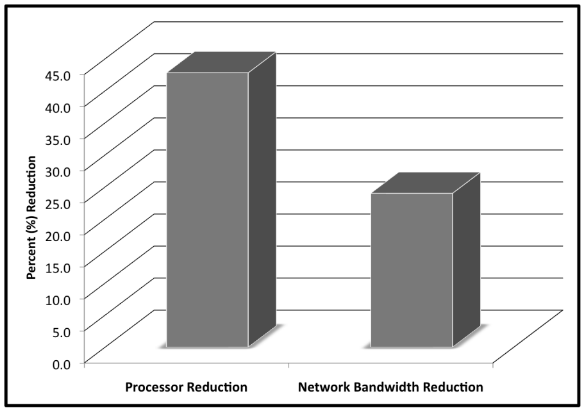

Figure 3. Network Bandwidth and Processor Reduction in Optimized Deployment

The ScatterD tool was also applied to minimize the bandwidth consumed due to communication by software tasks allocated to different processors in the production avionics system described in Section 3. Reducing the bandwidth requirements of the system leads to more efficient, faster communication while also reducing power consumption. The legacy deployment consumed $1.83 \cdot 10^{08}$ bytes of bandwidth. Both versions of the ScatterD tool yielded a deployment that reduced bandwidth by $4.39 \cdot 10^{07}$ or 24% as shown in Figure 3.

7 Concluding Remarks

Optimizing th deployment topologies on legacy embedded flight avionics system can yield substantial benefits, such as reducing hardware costs and emmission. By combining the efficiency of metaheuristic optimization techniques, such as particle swarm optimization, with other heuristic algorithms, such as bin-packing, legacy deployments can be evolved and optimized in a matter of seconds. The following are a summary of the lessons we learned applying our hybrid-heuristic bin-packing tool to optimize a legacy flight avionics system:

- **Multiple constraints make deployment planning hard**. Avionics deployments must adhere to a wide range of strict constraints, such as resource, collocation, scheduling, and network constraints. Deployment optimization technique must account for these constraints when determining a new deployment.

- **A Huge deployment space requires intelligent search techniques.** The vast majority of potential deployments that could be created violate one or more design constraints. Intelligent and automated techniques, such as hybrid-heuristic bin-packing, must therefore be applied to discover valid "near-optimal" deployments.

- **Substantial processor and network bandwidth reductions are possible.** Applying hybrid-heuristic bin-packing to the flight avionics system resulted in 42.8% processor reduction and 24% bandwidth reduction. Our future work is applying hybrid-heuristic bin-packing to other legacy embedded system deployments.

  The ScatterD tool is available in open source form int the Ascent Design Studio(`http://ascent-design-studio.googlecode.com`.) This problem was found on the SPRUCE web portal(`www.sprucecommunity.org`), a portal dedicated to pairing cutting edge industry challenge problems with appropriate expert researchers. A document describing the flight avionics system case study as well as additional details of our solutions are available at SPRUCE.

References

[1] H. Beitollahi and G. Deconinck. Fault-Tolerant Partitioning Scheduling Algorithms in Real-Time Multiprocessor Systems. *Pacific Rim International Symposium on Dependable Computing, IEEE*, 0:296–304, 2006.

[2] A. Bertossi, L. Mancini, and F. Rossini. Fault-Tolerant Rate-Monotonic First-Fit Scheduling in Hard-Real-Time Systems. *IEEE Transactions On Parallel and Distributed Systems*, pages 934–945, 1999.

[3] A. Burchard, J. Liebeherr, Y. Oh, and S. Son. New Strategies for Assigning Real-time Tasks to Multiprocessor Systems. *IEEE Transactions on Computers*, 44(12):1429–1442, 1995.

[4] A. Carzaniga, A. Fuggetta, S. Richard, D. Heimbigner, A. van der Hoek, A. Wolf, and COLORADO STATE UNIV FORT COLLINS DEPT OF COMPUTER SCIENCE. *A Characterization Framework for Software Deployment Technologies*. Defense Technical Information Center, 1998.

[5] W. Damm, A. Votintseva, A. Metzner, B. Josko, T. Peikenkamp, and E. Böde. Boosting Re-use of Embedded Automotive Applications Through Rich Components. *Proceedings of Foundations of Interface Technologies*, 2005, 2005.

[6] S. Lauzac, R. Melhem, and D. Mosse. Comparison of Global and Partitioning Schemes for Scheduling Rate Monotonic Tasks on a Multiprocessor. In *10th Euromicro Workshop on Real Time Systems*, pages 188–195, 1998.

[7] J. Stankovic. Strategic Directions in Real-time and Embedded Systems. *ACM Computing Surveys (CSUR)*, 28(4):751–763, 1996.