Modeling Multicomputer Task Allocation as a Vector Packing Problem

James E. Beck
Advanced Microcomputer Development
Delco Electronics Corporation

Daniel P. Siewiorek
Department of Electrical and Computer Engineering
Carnegie Mellon University

Abstract

This paper considers the problem of task allocation for embedded, bus–based multicomputers. The problem is shown to be isomorphic to a generalization of vector packing, and heuristic solution techniques are investigated. A total of 256 packing algorithms are considered, using a divide–and–conquer experimentation strategy on a set of sixteen real and synthetic test cases. Performance is compared based on the number of processors, the utilization level of the broadcast bus and run time. This research differs from other approaches in that task allocation is formulated as a "multi–dimensional" problem, and general purpose solution techniques are developed that can accommodate arbitrary models for the schedulable resources.

1 Introduction

The computing demands of today's embedded applications can exhaust the resources available on even the most sophisticated single processor systems. This has motivated interest in multiple processor architectures, and one alternative that is well–suited for embedded use is the *multicomputer* [1].

A multicomputer consists of a set of autonomous processing nodes that utilize a communication subsystem to interact with one another. Each node is autonomous in the sense that it has its own local address space and executes its own operating system (or none at all). Furthermore, the nodes can be heterogeneous, which allows the implementation to reflect the mix of special purpose hardware and signal processing required by the application.

Communication between nodes is coarse–grained and accomplished via message passing. In general, the communication subsystem is arbitrary, however a broadcast bus is often used because of its simplicity and low cost. The data link layer protocol of the bus is likewise arbitrary, but CSMA/CR protocols [14] (e.g. CAN [27]) are desirable because they provide reliable transmission of fixed–priority, real–time messages with guaranteed response times. This allows the bus to be treated as a schedulable resource. As such, it can be analytically modeled [28] and/or simulated [2] to determine timing correctness.

Given a multicomputer implementation of an embedded system, an important design consideration is how to partition the software modules across the hardware nodes. This is known as the *task allocation problem*. A solution entails finding an assignment of tasks to nodes such that each task is assigned to exactly one node and no node is over–utilized. Furthermore, task allocation needs to be carried out in a way that minimizes cost while satisfying the system's performance specifications. For embedded systems, many factors can influence task allocation decisions. In fact, task allocation decisions can be influenced by any design attribute that affects cost or performance. Accordingly, task allocation needs to be formulated as a *multi–dimensional problem*, and the tasks and nodes need to be modeled as *multi–dimensional objects*.

The task allocation problem is, in general, NP–complete, and heuristic algorithms are required. In this paper, the task allocation problem for bus–based multicomputers is shown to be isomorphic to a generalization of the vector packing problem. Solution techniques are developed by considering heuristic solutions to the packing problem. These techniques are presented, described and verified experimentally on a mixture of real and synthetic test cases.

The rest of this paper is organized as follows. Section 2 introduces a model of the task allocation problem. Section 3 re–states task allocation as a packing problem which is shown to be a generalization of vector packing. Section 4 reviews related work in the areas of packing and task allocation and describes the relevance of this work. Section 5 introduces heuristic algorithms for solving the packing problem. Section 6 describes the experimentation strategy and presents results obtained by applying the heuristic algorithms to both real and synthetic test cases. Section 7 concludes and summarizes the paper.

2 Modeling Task Allocation

This section presents an embedded system model. It serves as an abstraction for evaluating task allocation alternatives. For more detailed information, refer to [4].

2.1 Software Model

A synchronous data flow graph (SDFG) is used to model the software [16]. The nodes represent tasks and the arcs signify the communication between them. The amount of data produced and consumed on each task invocation is known a priori. Thus, the resource requirements of the application are statically predictable, allowing static task allocation.

A task's demand for hardware resources is application specific and can occur across many independent dimensions, such as CPU throughput, memory or I/O channels. An application specific *demand vector* is used to model this. This vector is of arbitrary length, and each element corresponds to a resource that is:

1. Available on one or more of the processing nodes.
2. A constraint on the solution.

Once the elements of the demand vector are defined, each task in the application is modeled by its own vector instance. Thus, the demand that a task imposes on a processing node is *multi–dimensional*, and defined with respect to the application specific demand vector.

2.2 Target Architecture

The target architecture consists of an arbitrary number of heterogeneous nodes that communicate via message passing over a bus. The resources available on a node are specified with respect to a *capacity vector*, which is analogous to the demand vector used to model the tasks.

To model the bus, an analytic function is used that predicts message transmission time as a function of message size. An example of such a function for the CAN bus is shown in Figure 1. This function is used in conjunction with a scheduling model (e.g. [28]) to determine real time schedulability of the bus.

$$C = \left\lceil \frac{D}{8} \right\rceil \frac{67}{h} + \frac{8D}{h}$$

C = Transmission Time (seconds)
D = Data Size (bytes)
h = CAN bus bit rate (bps)

Figure 1: CAN Message Format

2.3 Communication

If communicating tasks are assigned to different nodes, then message traffic will occur over the bus. However, since the bandwidth of the bus is finite, it can only accommodate a limited number of messages. Hence, a solution is feasible only if the cumulative message demand is less than the schedulable bandwidth of the bus.

Furthermore, the arcs within the SDFG define a precedence ordering among tasks. However, it is assumed that data traveling along the arcs is buffered, which de–couples the tasks and allows them to execute asynchronously at their own period, without violating precedence relationships. Thus, each processing node is presented with a periodic task set that is consistent with the assumptions required by rate monotonic scheduling [17, 18].

2.4 Feasibility Constraints

An assignment of tasks to nodes that satisfies all task requirements without over–utilizing any of the hardware components is said to be *feasible*. A set of *constraints* is needed to define the feasibility condition. These constraints are application specific, but they can be sorted into two broad categories: processor and bus constraints.

Processor constraints define when a node can support its task set. Likewise, bus constraints define when the bus can support a message set. These constraints are typically based on scheduling models for the processing nodes (e.g. [17, 18]) and bus (e.g. [2, 28]). The constraints used during experimentation are not shown here, but details can be found in [4].

2.5 Task Allocation Objectives

The goal of task allocation is to obtain an assignment of tasks to nodes that is *feasible*, and that;

1. Minimizes the number of processing nodes.
2. Minimizes the utilization level of the broadcast bus.

These are competing goals and they must be balanced effectively.

3 Task Allocation as a Packing Problem

Once a node is specified, only a subset of tasks can be assigned to it without violating feasibility constraints. This leads to the packing–based problem representation which is shown in Figure 2.

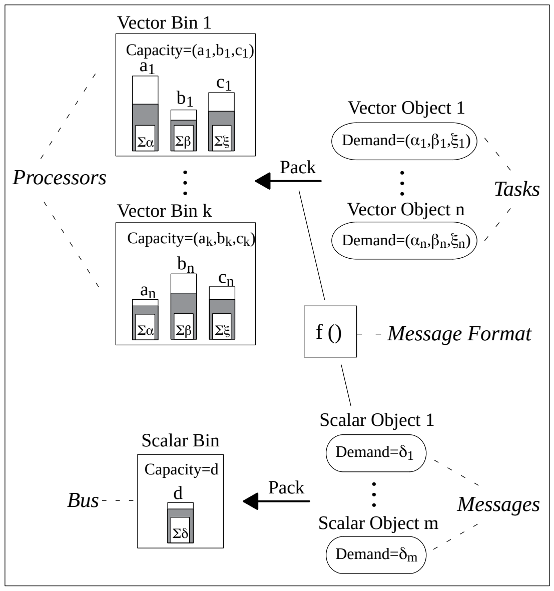

Figure 2: Packing–based Problem Model

Under this representation, each node is modeled by a *vector bin*. The bins are multi–dimensional, and the dimensions are defined by the capacity vector. For each bin, the capacity in each dimension is chosen to match the specifications of the processing node being modeled.

Similarly, each task is modeled by a *vector object*. The objects have demands for the hardware resources which are defined with respect to the demand vector. For each object, the demand for each resource is chosen to match the resource requirements of the task being modeled.

Lastly, the bus is modeled by a *scalar bin*. It is characterized by a scalar capacity, which is equal to the bus's schedulable bandwidth.

With this problem representation, task allocation becomes a matter of packing the vector objects into the vector bins such that none of the bins (including the bus's scalar bin) overflow. Note that demand for the bus's scalar bin represents the set of messages, which is determined as a by–product of task assignment decisions. Thus, the task allocation problem for bus–based multicomputers is isomorphic to the multi–dimensional packing problem illustrated in Figure 2.

When stated as a decision problem, the question is whether a set of vector objects can be packed into a set of vector bins such that none of the bins overflows, including the bus's scalar bin. Next, consider the special case where the scalar bin has infinite capacity[1]. In this situation, since there is no restriction on packing the scalar bin, it can be ignored and the problem reduces to the widely investigated *vector packing* problem. Furthermore, the packing problem shown in Figure 2 is NP–complete, since vector packing is NP–complete and the reduction described above is clearly a polynomial transformation [4].

A descriptive name for the new packing problem, based on its definition and relationship to vector packing, is *vector packing with constrained object groupings*, or **VPCOG**. This name will be used to refer to the problem in subsequent sections.

4 Related Work

The most prolific packing problem is *bin packing* [21]. Generalizing it to n–dimensions leads to *multi–dimensional bin packing*, and there are two variants of this: *rectangle packing* [8] and *vector packing* [12]. Rectangle packing is a geometric problem, where a set of n–dimensional polyhedra are packed into n–dimensional bins. The polyhedra assigned to a bin must fit within it without intersecting one another. This is easily visualized in n=3 dimensions where it is analogous to the *knapsack problem* [21]. Conversely, with vector packing, the n–dimensions are independent. The objects are packed into bins with the restriction that the vector sum of the object sizes cannot exceed the bin's vector capacity. VPCOG, the packing problem described in this paper, is a generalization of vector packing that restricts object groupings. Figure 3 illustrates the proper perspective between VPCOG and other packing problems.

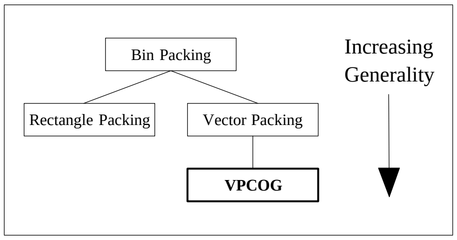

Figure 3: Hierarchy of Packing Problems

Task allocation can either be done statically or dynamically. Static techniques are limited to applications with predictable run–time behavior, which applies to the class of design problems considered by this research. Previous approaches to static task allocation can be loosely divided into three categories: graph theoretic [5, 19, 23, 26], mathematical programming [6, 20] and heuristic [7, 9, 11, 25].

The allocation techniques described in this paper are based on heuristic solutions to the VPCOG packing problem. They differ from existing approaches in two main regards. First, tasks and nodes are modeled as multi–dimensional objects, and task allocation is treated as a multi–dimensional problem. Second, these algorithms base task assignment decisions on a set of user–supplied feasibility constraints[2]. These constraints are arbitrary, and they allow the designer to incorporate precise scheduling models for the hardware resources (e.g. CPU and bus). Thus, the solutions obtained by the allocation algorithms are provably correct within the context of the system's timing requirements.

5 Heuristic Packing Algorithms

This section describes heuristic solutions to the VPCOG packing problem. A set of candidate algorithms are presented, inspired by the classic first– and best–fit solutions to the bin packing problem. All of the algorithms are *one–pass* and *greedy*. They progress by choosing objects, one by one, and assigning them to bins. This continues until all objects are assigned and the packing is complete, or else a set of objects remain that will not fit into any of the bins. In this case, the algorithm fails.

A total of 256 algorithms are considered. They are specified with respect to a five–character acronym which is defined in Figure 4. The first character of the acronym specifies the bin selection method. Likewise, the second character defines how the utilization level of a vector bin is measured. If two or more bins are found to be equally good according to the bin selection policy, then a tie–breaking strategy is needed. The third character specifies the tie–breaking criterion. The fourth character specifies the order in which the vector objects are packed. The fifth and final character of the acronym specifies the method for determining the size of a vector object.

6 Results

A set of experiments were performed to gauge the effectiveness of the heuristic packing algorithms. The instance of the VPCOG problem shown in Figure 5 was used during experimentation. The problem requires tasks to be packed into *unit processing elements* which communicate over a 1 Mbps CAN bus. Intra–processor communication is free, while inter–processor communication consumes bus bandwidth (based on the function shown in Figure 1). The capacity and demand vectors contain six elements, which correspond to CPU throughput, ROM, RAM, digital I/O channels, analog I/O channels, and pulse–width modulated timer channels, respectively. The unit processing element's resource capacities were chosen arbitrarily. Tasks have varying levels of demand for the resources. Demand is computed statically, and is known prior to allocation. To determine feasibility, resource capacities are

1. This corresponds to the hypothetical case of a bus with unlimited schedulable bandwidth.

2. Within the packing paradigm, the feasibility constraints are invoked to determine whether a vector object will "fit" inside of a bin, without causing any of the bins to "overflow".

compared against the cumulative demand imposed by the task (and message) sets.

Sixteen real and synthetic SDFGs were used as test cases. The real test cases were adapted from a commercial automotive electronic application described in [3]. The synthetic test cases are a mixture of random and hand generated SDFGs which, taken as a whole, span a large portion of the design space.

When a heuristic packing algorithm is applied to a test case, three metrics are used to gauge its effectiveness:

1. Number of vector bins (i.e. unit processors) used.
2. Scalar bin (i.e. bus bandwidth) utilization level.
3. Run time

Experiments were carried out in four stages, and a divide–and–conquer method was used to compare the 256 possible algorithms. The results are summarized in the Appendix, and the subsections that follow describe the stages.

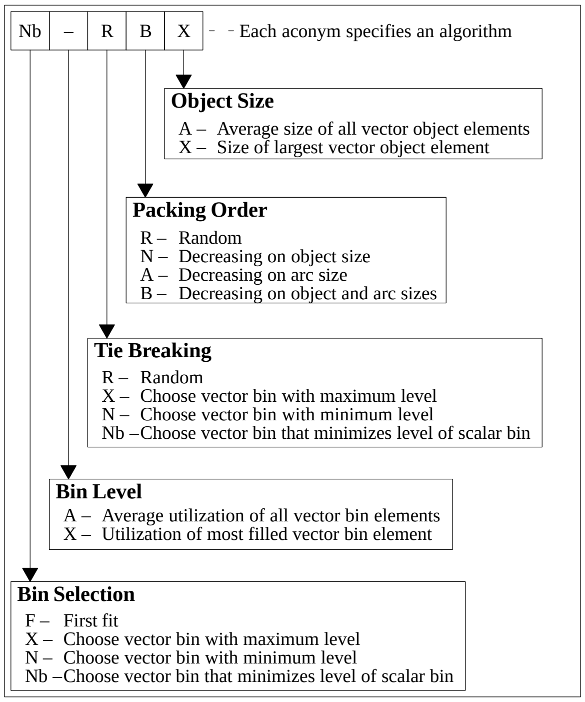
Nb
–
R
B
X
Each aconym specifies an algorithm
Object Size
A – Average size of all vector object elements
X – Size of largest vector object element
Packing Order
R – Random
N – Decreasing on object size
A – Decreasing on arc size
B – Decreasing on object and arc sizes
Tie Breaking
R – Random
X – Choose vector bin with maximum level
N – Choose vector bin with minimum level
Nb – Choose vector bin that minimizes level of scalar bin
Bin Level
A – Average utilization of all vector bin elements
X – Utilization of most filled vector bin element
Bin Selection
F – First fit
X – Choose vector bin with maximum level
N – Choose vector bin with minimum level
Nb – Choose vector bin that minimizes level of scalar bin

Figure 4: Candidate Packing Algorithms

6.1 Stage One

The goal of stage one is to determine the effect of node ordering. This is done by comparing first–fit algorithms with different ordering schemes. Specifically, the following six algorithms are considered:

F—R–(baseline)[3]	F—NX	F—NA
F—A–	F—BX	F—BA

The results show that ordering based on object size tends to decrease the number of vector bins, but results in poor utilization of the scalar bin. In fact, this prevents a feasible solution from being found for three of the test cases. Conversely, ordering on arc size tends to decrease the scalar bin utilization level, but often requires more vector bins than the baseline. Ordering on both object and arc sizes works best, outperforming all other schemes. This method exploits the benefits of object ordering and arc ordering (i.e. fewer vector bins and lower scalar bin utilization) without suffering from their weaknesses (i.e. not finding a solution). The effect of basing node size on the size of the maximum or average element was marginal and inconclusive. Furthermore, the run time variation across algorithms was not significant, since ordering the objects only requires them to be sorted once (O(nlgn)) before packing begins.

3. As defined in Figure 4, (F—R–) = First –fit bin selection and random node ordering.

To summarize, ordering based on object and arc sizes is the most effective technique, and equating object size to the size of the maximum vector element is preferred for simplicity. Hence, only (xxxBX) algorithms are considered in subsequent stages.

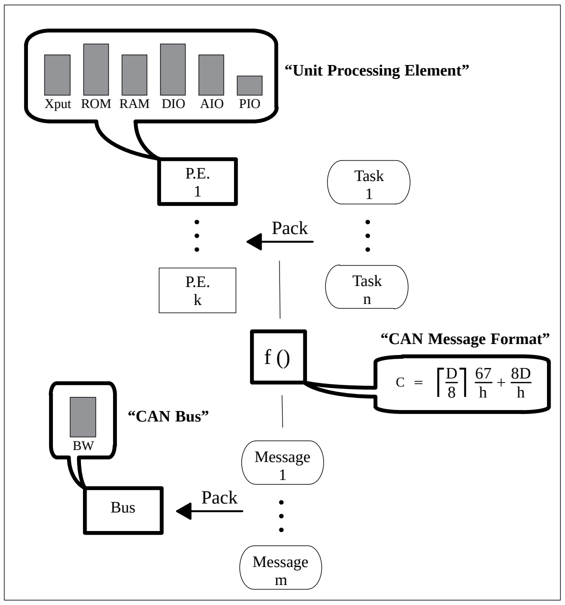

Figure 5: VPCOG Instance used in Experiments

6.2 Stage Two

The goal of stage two is to investigate the effect of the bin selection policy. This is accomplished by comparing the following six algorithms:

F—BX (baseline)[4]	XXRBX	XARBX
NXRBX	NARBX	Nb–RBX

The results reveal several things. First, selecting the least utilized vector bin (i.e. NxRBX) performs worse than the first–fit baseline algorithm (F—BX). This is intuitive: Selecting the least utilized bin is a poor packing heuristic since it never encourages completely filling a started bin. This leads to excessive resource fragmentation and more bins, on average, than first–fit. Likewise, selecting the most utilized vector bin (i.e. XxRBX) requires no fewer vector bins, on average, than the first–fit

4. As defined in Figure 4, (F—BX) = First –fit bin selection, decreasing order based on object and arc sizes, and object size based on the maximum vector element.

baseline algorithm. This underscores the fact that vector packing has no analog to bin packing's *best–fit–decreasing* algorithm. The reason for this is also intuitive: There is no notion of what constitutes a "best fit" for a vector object being placed into a vector bin. In fact, the bin that is the "best fit" for any particular dimension may actually be a poor choice across the remaining dimensions [4].

Selecting the vector bin that minimizes utilization of the scalar bin (i.e. Nb–RBX) out–performs the other techniques, including the first–fit baseline algorithm, (F—BX). On average, this technique requires no more or less vector bins than first–fit, but it does yield significantly lower scalar bin (i.e. bus bandwidth) utilization levels. In fact, this method is a good choice for task allocation since, coupled with the object ordering scheme found in stage one, it leads to a natural and dynamic clustering of heavily communicating tasks.

The baseline algorithm, (F—BX), does have a run time advantage over the other algorithms. However, because of its superior solution qualities, (Nb–RBX) is preferred and only (NbxxBX) algorithms are considered in the experimentation stages that follow.

6.3 Stage Three

The goal of stage three is to determine the effect of employing a tie breaking strategy. To accomplish this, the following two algorithms are compared:

Nb–RBX (baseline)   NbXXBX

As the results indicate, tie–breaking produces no measurable benefit.

6.4 Stage Four

The goal of the fourth and final stage is to compare the best heuristic algorithm (Nb–RBX) against optimum solutions. Optimum solutions are obtained by using two exhaustive search algorithms. The first search algorithm, OPT–BINS, returns the solution with the fewest vector bins, and it has a time complexity of $O({Bins}^{Objects})$. The second, OPT–BUS, returns the solution with the minimum scalar bin utilization, and it has a time complexity of $O(2^{Arcs})$. Because of the large time complexities of the search algorithms, it is only practical to apply them to the two smallest test cases. The results are summarized in Figure 6.

	Syn12		Syn13	
Algorithm	Bins	Bus Util.	Bins	Bus Util.
OPT–BINS	4	0.31	3	0.40
OPT–BUS	5	0.28	5	0.22
Nb–RBX	4	0.28	4	0.40

Figure 6: Nb–RBX vs. Optimum

(Nb–RBX) clearly affects a balance between the competing goals of minimizing the number of bins and minimizing utilization of the scalar bin (i.e. bus). Furthermore, since the time complexity of (Nb–RBX) is $O(bins \cdot objects)$, it has an obvious run time advantage over the search algorithms. In fact, (Nb–RBX) had run times which were several orders of magnitude faster for the test cases shown in Figure 6.

7 Summary

This paper described a generalization of the vector packing problem, VPCOG, which was shown to be isomorphic to task allocation for bus–based multicomputers. Since task allocation and the corresponding packing problem are NP–complete, heuristic solution techniques are required. A total of 256 heuristic packing algorithms were considered, and their performance was compared using a divide–and–conquer experimentation method on sixteen real and synthetic test cases with respect to three metrics: the number of vector bins (i.e. processing nodes), the utilization level of the scalar bin (i.e. bus bandwidth) and run time. Through experimentation, the (Nb–RBX) algorithm was found to be the most effective heuristic.

The (Nb–RBX) packing algorithm represents an effective and efficient way of performing task allocation for bus–based multicomputers. It is capable of minimizing the number of processing nodes needed for a design while simultaneously minimizing the utilization level of the broadcast bus. Furthermore, it supports a multi–dimensional representation of the task allocation problem, and it allows scheduling models to be incorporated so that timing correctness is achieved as a by–product of task allocation.

8 References

[1] W. Athas and C. Seitz. Multicomputers: Message–Passing Concurrent Computers. *Computer*, 21(8):9–23, Aug. 1988.

[2] M. Baba and E. Powner. Scheduling Performance in Distributed Real–Time Control System, In *Proc. of the 2nd Int. CAN Conf.*, pp. 2–11, Sept. 1995.

[3] J. Beck. *Characterization of an automotive powertrain control application.* Tech. Report, Delco Electronics Corp., May 1994.

[4] J. Beck. *Automated Processor Specification and Task Allocation Methods for Embedded Multicomputer Systems.* Ph.D. Thesis, Carnegie Mellon Univ., Apr. 1995.

[5] F. Berman and L. Snyder. On Mapping Parallel Algorithms into Parallel Architectures. In *Proc. of the Int. Conf. on Parallel Processing*, pp. 307–309, 1984.

[6] A. Billionnet, M. Costa and A. Sutter. An Efficient Algorithm for a Task Allocation Problem. *Journal of the Association for Computing Machinery,* 39(3):502–518, Jul. 1992.

[7] W. Chu and L. Lan. Task Allocation and Precedence Relations for Distributed Real–Time Systems. *IEEE Trans. on Computers,* 36(6):667–679, Jun. 1987.

[8] D. Coppersmith and P. Raghavan. Multidimensional on–line bin packing: algorithms and worst–case analysis, *Operations Research Letters,* 8(1):17–19, Feb. 1989.

[9] K. Efe. Heuristic Models of Task Assignment Scheduling in Distributed Systems. *Computer,* 15(6):50–56, Jun. 1982.

[10] J. Gaudiot, J. Pi and M. Campbell. Program graph allocation in distributed multicomputers. *Parallel Computing,* 7(2):227–247, Jun. 1988.

[11] C. Houstis. Module Allocation of Real–Time Applications to Distributed Systems. *IEEE Trans. on Software Engineering,* 16(7):699–709, Jul. 1990.

[12] R. Karp, M. Luby and A. Marchetti–Spaccamela. A probabilistic analysis of multidimensional bin packing problems. In *Proc. of the annual ACM symp. on theory of computing,* pp. 289–298, 1984.

[13] C. Koutsougeras, C. Papachristou and R. Vemuri. Data Flow Graph Partitioning to Reduce Communication Cost. In *Proc. of the 19th Annual Workshop on Microprogramming,* pp. 82–91, Oct. 1986.

[14] J. Kurose, M. Schwartz and Y. Yemini. Multiple–Access Protocols and Time–Constrained Communication. *Computing Surveys* 16(1):43–70, Mar. 1984.

[15] C. Lee, M. Kim and C. Park. An efficient k–way graph partitioning algorithm for task allocation in parallel computing systems. In *Proc. of the First Int. Conf. on Systems Integration*, pp. 748–751, Apr. 1990.

[16] E. Lee and and D. Messerschmitt. Synchronous Data Flow. *In Proc. of the IEEE* (75)9:1235–1245, Sep. 1987.

[17] J. Lehoczky, L. Sha and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proc. of the IEEE Real–Time Systems Symposium*, pp. 166–171, Dec. 1989.

[18] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real–time environment. *Journal of the ACM* 20(1):46–61, Jan. 1973.

[19] V. Lo. Heuristic Algorithms for Task Assignment in Distributed Systems. In *Proc. of the Int. Conf. on Distributed Computing Systems*, pp. 30–39, 1984.

[20] P. Ma, E. Lee and M. Tsuchiya. A Task Allocation Model for Distributed Computing Systems. *IEEE Trans. on Computers*, 31(1):41–47, Jan. 1982.

[21] T. Nielsen. *Combinatorial Bin Packing Problems*. University of Arizona, 1985.

[22] T. Ravi. *Partitioning and Allocation of Functional Programs for Data Flow Processors*. Tech Report CSD–860063, UCLA, Apr. 1986.

[23] J. Ryou and J. Wong. *A Heuristic Algorithm for Task Allocation in Distributed Computer Systems*. Tech. Report TR 88–7, Iowa State Univ.., 1988.

[24] T. Saponas. A Real–Time Distributed Processing System. In *Proc. of the IEEE Real–Time Systems Symposium*, pp. 36–43, 1986.

[25] B. Shirazi and M. Wang. *Evaluation of Three Heuristic Functions for Task Allocation*. Tech. Report 87–CSE–28, Southern Methodist Univ., 1987.

[26] H. Stone. Multiprocessor Scheduling with the Aid of Network Flow Algorithms. *IEEE Trans. on Software Engineering*, 3(1):85–93, Jan. 1977.

[27] C. Szydlowski. The CAN Specifications. In *Proc of the Fifth Annual Embedded Systems Conf.*, pp. 365–374, Oct. 1993.

[28] K. Tindell and A. Burns. Guaranteeing Message Latencies on Control Area Network (CAN), In *Proc. of the 1st Int. CAN Conf.*, pp. 2–11, Sept. 1994.

Appendix

Run Times (Seconds)

Alg	Syn01	Syn02	Syn03	Syn04	Syn05	Syn06	Syn07	Syn08	Syn09	Syn10	Syn11	Syn12	Syn13	Real01	Real02	Real03
F—R–	12.8	0.7	<0.1	41.5	1.9	293.1	0.5	8.4	10.1	0.7	34.3	<0.1	<0.1	2.3	3.5	0.3
F—NX	4.7	0.6	<0.1	18.2	2.6	131.6	0.4	5.0	5.7	0.8	17.3	<0.1	<0.1	1.4	2.4	0.3
F—NA	5.5	0.6	<0.1	26.7	1.8	162.1	0.5	7.0	7.0	0.8	15.6	<0.1	<0.1	206.8	4.6	0.2
F—A–	14.5	0.7	<0.1	41.0	0.1	285.2	0.8	12.9	15.2	0.1	44.2	<0.1	<0.1	3.7	8.1	0.4
F—BX	11.5	0.7	<0.1	17.8	0.1	145.9	0.6	8.3	8.3	0.1	17.3	<0.1	<0.1	2.9	5.1	0.3
F—BA	10.1	0.8	<0.1	32.2	0.1	205.9	0.6	8.5	9.0	0.1	18.2	<0.1	<0.1	2.8	4.1	0.3
XXRBX	23.7	1.3	<0.1	31.5	0.1	263.5	1.0	14.8	16.2	0.1	34.7	<0.1	<0.1	5.3	12.4	0.5
XARBX	23.4	1.3	<0.1	31.7	0.1	263.5	1.2	15.3	16.2	0.1	34.9	<0.1	<0.1	5.3	12.6	0.6
NXRBX	24.5	1.5	<0.1	34.5	0.1	251.3	2.3	14.6	17.6	0.3	62.1	<0.1	<0.1	5.8	13.7	0.7
NARBX	29.4	1.7	<0.1	43.0	12.6	309.6	2.0	22.1	23.8	0.3	81.4	<0.1	<0.1	6.8	17.6	1.0
Nb–RBX	22.0	1.3	<0.1	31.6	0.1	267.3	1.0	15.0	15.9	0.1	33.5	<0.1	<0.1	5.7	12.2	0.6
NbXXBX	23.5	1.7	<0.1	34.1	0.1	284.6	1.1	16.2	17.3	0.1	36.3	<0.1	<0.1	5.5	12.9	0.5

Number of Vector Bins (Normalized to the lower bound)

Alg	Syn01	Syn02	Syn03	Syn04	Syn05	Syn06	Syn07	Syn08	Syn09	Syn10	Syn11	Syn12	Syn13	Real01	Real02	Real03
F—R–	1.063	1	–	1.115	–	1.092	1	1.097	1.083	–	1.387	1.333	1.333	1	1	1
F—NX	1	1	–	1.033	–	1.020	1	1.032	1	–	1.224	1.333	1	1	1	1
F—NA	1.031	1	–	1.049	–	1.020	1	1.065	1.028	–	1.184	1.333	1.333	–	1.067	1
F—A–	1.063	1	1	1.115	1	1.071	1	1.097	1.083	1	1.306	1.333	1.333	1	1.067	1
F—BX	1.031	1	1	1.033	1	1.031	1	1.065	1.055	1	1.245	1.333	1.333	1	1.067	1
F—BA	1.031	1	1	1.049	1	1.031	1	1.097	1.055	1	1.224	1.333	1	1	1	1
XXRBX	1.031	1	1	1.033	1	1.020	1	1.065	1.055	1	1.245	1.333	1.333	1	1.067	1
XARBX	1.031	1	1	1.033	1	1.020	1	1.065	1.055	1	1.224	1.333	1.333	1	1.067	1
NXRBX	1.125	1	1	1.066	1	1.041	1.300	1.129	1.139	1.4	1.510	1.333	1.333	1	1.067	1
NARBX	1.094	1	1	1.049	–	1.041	1.100	1.129	1.139	1.4	1.551	1.667	1.667	1	1.067	1
Nb–RBX	1.031	1	1	1.033	1	1.031	1	1.065	1.055	1	1.224	1.333	1.333	1	1.067	1
NbXXBX	1.031	1	1	1.033	1	1.031	1	1.065	1.055	1	1.224	1.333	1.333	1	1.067	1

Scalar Bin (i.e. Bus Bandwidth) Utilization

Alg	Syn01	Syn02	Syn03	Syn04	Syn05	Syn06	Syn07	Syn08	Syn09	Syn10	Syn11	Syn12	Syn13	Real01	Real02	Real03
F—R–	0.2715	0.9084	–	0	–	0.1283	0.9101	0.5326	0.2341	–	0.9288	0.4153	0.7010	0.8760	0.8503	0.2341
F—NX	0.2752	0.8627	–	0	–	0.1244	0.9320	0.5109	0.2390	–	0.9288	0.6318	0.4000	0.9550	0.9461	0.3291
F—NA	0.2687	0.9003	–	0	–	0.1283	0.9372	0.5213	0.2349	–	0.9288	0.7280	0.3754	–	0.9821	0.3347
F—A–	0.2775	0.7609	0.8625	0	0.8882	0.1244	0.8320	0.5109	0.2341	0.6978	0.9288	0.4153	0.4079	0.6641	0.6287	0.1378
F—BX	0.2763	0.7836	0.8625	0	0.8882	0.1244	0.8320	0.5109	0.2341	0.6978	0.9288	0.4153	0.4071	0.6641	0.6287	0.1378
F—BA	0.2681	0.7609	0.8625	0	0.8882	0.1244	0.8769	0.5215	0.2350	0.6978	0.9288	0.4153	0.3797	0.6123	0.5972	0.1831
XXRBX	0.2657	0.7609	0.8625	0	0.8882	0.1168	0.8487	0.5042	0.2390	0.6978	0.9288	0.4153	0.2321	0.6856	0.6287	0.1378
XARBX	0.2710	0.7757	0.8625	0	0.8882	0.1134	0.8320	0.5092	0.2338	0.6978	0.8367	0.4153	0.4006	0.6588	0.6290	0.1378
NXRBX	0.2792	0.9072	0.9145	0	0.9898	0.1283	0.9254	0.5323	0.2390	0.9993	0.9288	0.7280	0.5246	0.9845	0.9648	0.3368
NARBX	0.2760	0.9964	0.9145	0	–	0.1283	0.9531	0.5274	0.2342	0.9958	0.9288	0.6318	0.5549	0.9981	0.9696	0.3298
Nb–RBX	0.2262	0.7430	0.8625	0	0.8882	0.1210	0.7712	0.5042	0.2205	0.6978	0.8367	0.2863	0.4007	0.6550	0.6090	0.1377
NbXXBX	0.2262	0.7430	0.8625	0	0.8882	0.1210	0.7712	0.5042	0.2205	0.6978	0.8367	0.2863	0.4007	0.6550	0.6090	0.1377

"–" indicates that the algorithm failed to find a solution.