# A Dynamic Programming Approach to Task Partitioning upon Memory-constrained Multiprocessors*

Sanjoy Baruah and Nathan Fisher

Department of Computer Science
The University of North Carolina

**Abstract.** Most prior theoretical research on partitioning algorithms for real-time multiprocessor platforms has focused on ensuring that the cumulative computing requirements of the tasks assigned to each processor does not exceed the processor's processing power. However, many multiprocessor platforms have only limited amounts of local per-processor memory; if the memory limitation of a processor is not respected, thrashing between "main" memory and the processor's local memory may occur during run-time and may result in performance degradation. The *memory constrained multiprocessor partitioning problem* — the problem of task partitioning in a manner that is cognizant of both memory and processing capacity constraints — is known to be NP-hard in the strong sense. In this work, we prove that a restricted version of this problem, in which there are only a few distinct *types* of tasks and all the tasks invoked during runtime are instantiations of these few types, is tractable, and present a polynomial-time algorithm for solving it.
**Keywords:** Multiprocessor systems; Partitioned scheduling; Memory-constrained systems; Dynamic programming.

## 1 Introduction

As the functionality demanded of real-time embedded systems has increased, it is becoming unreasonable to expect to implement them upon uniprocessor platforms; hence, multiprocessor platforms are increasingly used for implementing such systems. Efficient system implementation on such multiprocessor platforms may require the careful management of several key resources, such as processor capacity, memory capacity, communication bandwidth, etc. For instance, in assigning tasks to processors care must be taken to ensure that both the limited computing capacity of a processor *and* its limited local memory is taken into consideration.

Consider as an example programmable *network processors* (NPs) which are proving increasingly popular for implementing network switches and routers; in

* Supported in part by the National Science Foundation (Grant Nos. ITR-0082866, CCR-0204312, and CCR-0309825).

particular, the IXP 2800, which is the high-end member of the Intel IXP family of NPs. NPs offer an alternative to conventional ASIC (application-specific integrated circuit) designs for deploying customized functions, such as firewalls, intrusion detection, load balancing, virtual private networks, protocol conversions, etc., in switches and routers on the Internet. Among the benefits of programmable NPs over ASIC designs is that NPs allow such custom functions to be deployed in a rapid and cost-effective manner; as the Internet continues to evolve at a rapid rate, such NPs are expected to become important building blocks for network elements in all parts of the network, from the edge to the core. The IXP 2800 contains an XScale (ARM architecture) core processor, sixteen independent RISC CPUs called *microengines* (MEs), interface controls for access to off-chip SRAM and DRAM, and standard interfaces to media or a switch fabric. The 32-bit XScale core processor is intended for use for control functions such as managing routing tables or other state information, or handling exception packets. Each 32-bit ME has access to private storage for 4K instructions and 640 words of local memory: these MEs are intended for creating multiple parallel task pipelines for performing (perhaps different) processing functions on multiple packets concurrently. In mapping tasks to these MEs it is necessary that, in addition to not overloading the ME's computing capacity, the total code-size of all tasks assigned to a particular ME not exceed 4K instructions. (Technical specifications of the IXP 2800 are available at `http://www.intel.com/design/network/products/npfamily/ixp2800.htm`.)

Most prior theoretical research on partitioning algorithms for real-time multiprocessor platforms has focused on ensuring that the cumulative computing requirements of the tasks assigned to each processor does not exceed the processor's computing capacity [1, 2]. We have recently been conducting research that can be considered to be a generalization of this earlier work, in the sense that it is aimed at determining strategies for assigning tasks to processors in multiprocessor platforms in which *several resources are only available in limited amounts on each processor*. In [3], we considered systems in which there are two such constraining resources – *local memory* for storing program code, and *computation capacity*. Given a multiprocessor comprised of several processors, each with its own (limited) processing capacity and local memory, and a collection of tasks, each characterized by its code-size and its computation requirement, the *memory-constrained multiprocessor partitioning problem* attempts to partition the tasks among the processors such that neither the memory capacity, nor the computing capacity on any processor is exceeded. We proved that this problem is intractable, and obtained a *sufficient* (albeit not necessary) condition for determining whether a given collection of tasks could be efficiently partitioned on a given multiprocessor system (and if so, an efficient algorithm for actually partitioning the tasks).

In this paper, we consider a restricted version of the general memory-constrained multiprocessor partitioning problem. In this restricted version, there are only a relatively small, constant, number of distinct *kinds* of tasks, and all the tasks that need to be partitioned are multiple instantiations of these few distinct

kinds of tasks. This restricted problem version accurately models, for example, the situation in network processors (NPs) in which there are a few different kinds of functionalities supported by an NP but the number of instances of tasks corresponding to each functionality may differ depending upon the current/ anticipated traffic through the NP. For this problem, we design an exact algorithm based upon the technique of dynamic programming that successfully partitions all systems that can be partitioned, and that runs in time polynomial in the number of tasks being partitioned and the number of processors comprising the platform.

The remainder of this paper is organized as follows. In Section 2, we introduce the general problem of partitioning tasks on memory-constrained multiprocessor platforms, show that it is intractable, and briefly list related research. In Section 3, we formally define the restricted version of the problem that is briefly described in the previous paragraph, and present a polynomial-time algorithm for solving this restricted problem. We conclude in Section 4 by placing the results here within a larger context of related eork on multiprocessor scheduling, and by briefly listing interesting questions that remain to be addressed.

## 2 The general problem, and related research

We consider the problem of mapping a given collection of tasks upon a platform comprised of multiple processors. We assume that all processors are *identical*, in the sense that they have exactly the same computing capacity and the same amount of local memory available.

A **task** $i$ is characterized by two parameters

- its *utilization* $u_i$, denoting the fraction of the computing capacity of a single processor that must be reserved for executing it; and
- its *code-size* $s_i$, denoting the fraction of the local memory associated with a single processor that must be reserved for storing its program code.

(Note that we make no assumptions about the relationship between $u_i$ and $s_i$ for a task $i$: in particular, we do not require that tasks $i$ with small $u_i$ have small $s_i$, and those with large $u_i$ have large $s_i$. Such restrictions would not allow us to model, e.g., a relatively simple task that is extremely computation-intensive because it repeatedly samples external input at a rapid rate, or a task with large code-size, comprised of much conditional code, that is invoked very infrequently.)

**Definition 1 (Memory-constrained multiprocessor partitioning problem).** ***Given*** *a collection of tasks* $\{1, 2, \ldots, n\}$ *and a number* $m$ *of processors* ***determine*** *a mapping function* $\chi : \{1, \ldots, n\} \rightarrow \{1, \ldots, m\}$ *such that the following* $2m$ *conditions are satisfied:*

$$\textit{for all } j, 1 \leq j \leq m, \left( \sum_{\textit{all } i \ | \chi(i)=j} u_i \leq 1, \right)$$

$$\textit{for all } j, 1 \leq j \leq m, \left( \sum_{\textit{all } i \ | \chi(i)=j} s_i \leq 1 \right)$$

□

It turns out that this problem is, in fact, intractable:

**Theorem 1 (From [3]).** *The memory-constrained multiprocessor partitioning problem is NP-complete in the strong sense.*

**Related research.** When either the utilization limitation or the memory limitation may be ignored, task partitioning is essentially a bin-packing problem: Each processor is a "bin" of capacity one, and each task assigned to it consumes an amount of this capacity equal to its utilization/code-size. While bin-packing is known to be NP-complete in the strong sense, efficient approximation algorithms and polynomial-time approximation schemes are known [4, 5] that can be used to determine task assignments with behaviour that is bounded in the worst-case. Unfortunately, when both utilization and memory limitations are considered simultaneously, the task assignment problem becomes much more difficult. This problem bears remarkable similarities to the $M$-Dimensional Vector Packing Problem ($M$-DVPP) [6] with $M = 2$. In particular, the tasks can be modeled as two-dimensional vectors (the two dimensions correspond to the code-size and utilization requirements, respectively), and the processors as bins that are characterized by two distinct capacities, memory size and computing capacity.[1]

The $M$-DVPP, like bin-packing, is known to be intractable (NP-hard in the strong sense [6]). Unfortunately, unlike bin-packing, for which even simple heuristics (such as best-fit, first-fit, etc. [4, 5]) have good worst-case performance guarantees, good polynomial-time approximation algorithms for $M$-DVPP provably cannot exist [7] (see also [8]), although several greedy heuristic algorithms have been studied in the literature [9, 6, 10, 11]. In particular, Beck and Siewiorek [12] have experimentally evaluated several vector packing heuristics for task allocation: while some heuristics were able to obtain acceptable allocations for certain specific kinds of task systems, there seems to be no heuristic that consistently (and provably) comes up with near-optimal allocations.

In addition to the research described above, there is much prior work on multiprocessor task scheduling and allocation problems based upon heuristic approaches. Algorithms have been proposed based on genetic algorithms [13], constraint logic programming [14, 15], and other heuristic approaches [16]; while it may be of some interest to determine whether such approaches are applicable for the memory-constrained multiprocessor partitioning problem, this is not within the scope of the current paper.

[1] The $M$-DVPP differs from the well-known $M$-dimensional bin-packing problem ($M$-DBPP) in that the $M$ different dimensions in $M$-DVPP are independent of one another while the $M$-DBPP concerns the packing of hyper-rectangles into hyper-cubes.

## 3 The restricted problem

We have seen above (Theorem 1) that the general problem of memory-constrained multiprocessor partitioning is intractable. In this section, we consider a more restricted version of this problem, for which we are able to determine a polynomial-time solution. This restricted problem models those application instances for which there are only a few kinds or *types* of tasks, and all the tasks that get instantiated during run-time are merely multiple instances of these task types.

**Definition 2.** *We start with some definitions, notation, and terminology, which will be used in the remainder of this paper.*

- *We assume that there are $k$ distinct* **types** *of tasks for some* constant $k$*; each type-$\ell$ task has a utilization $u_\ell$ and a memory-requirement $s_\ell$, for $1 \leq \ell \leq k$. These $k$ utilizations and memory requirements will be denoted by the vectors $\boldsymbol{u}_k$ and $\boldsymbol{s}_k$ respectively.*
- *A* **task-set specification** *(or simply, a* task specification*), is a vector $\boldsymbol{n}_k = (n_1, n_2, \ldots, n_k)$ of non-negative integers, and denotes that there are $n_\ell$ type-$\ell$ tasks; $1 \leq \ell \leq k$.*
- *A* **system** *is specified by a 4-tuple $(\boldsymbol{u}_k, \boldsymbol{s}_k, \boldsymbol{n}_k, m)$ where $\boldsymbol{u}_k$ and $\boldsymbol{s}_k$ denote the utilization and memory requirements respectively of the $k$ types of tasks, and task specification $\boldsymbol{n}_k$ a collection of $\sum_{\ell=i}^{k} n_\ell$ tasks that are to be implemented upon an $m$-processor platform.*
- *Task specification $\boldsymbol{n}_k$ is said to be $m$-***feasible** *if the tasks in $\boldsymbol{n}_k$ can be partitioned among $m$ processors such that each task's utilization and memory requirements are satisfied.*

**The problem** that is addressed in the remainder of this paper is the following: ***Given*** multiprocessor system $(\boldsymbol{u}_k, \boldsymbol{s}_k, \boldsymbol{n}_k, m)$, ***determine*** whether the task specification $\boldsymbol{n}_k$ is $m$-feasible.

**Notions of feasibility.** At least two distinct notions of partitioned feasibility can be defined for such systems upon multiprocessor platforms, depending upon the nature and the semantics of the tasks:

1. Each task mapped on to a processor needs its *own* copy of the program code; hence, each type-$\ell$ task needs to to have available for its exclusive use $u_\ell$ units of computing capacity, and $s_\ell$ units of memory, on the processor to which it is assigned.
2. Code is reentrant; hence, all type-$\ell$ tasks assigned to a processor can share a single copy of the code. While each type-$\ell$ task needs exclusive access to $u_\ell$ units of computing capacity on the processor to which it is assigned, *all* type-$\ell$ tasks assigned to a processor together need $s_\ell$ units of memory.

In the remainder of this section we will consider both of these notions of feasibility, and will design algorithms for determining feasible task mappings under both notions.

### 3.1 A feasibility analysis algorithm

Our approach is based upon *dynamic programming*: for any system $(\boldsymbol{u}_k, \boldsymbol{s}_k, \boldsymbol{n}_k, m)$, we construct a **feasibility table**. Each column in the feasibility table corresponds to a task specification that is a different subset of the set of tasks $\boldsymbol{n}_k$; hence, there are $\left(\Pi_{\ell=i}^{k}(n_\ell + 1)\right) = \mathcal{O}(n^k)$ columns. The table has $m$ rows, and it will be filled such that the entry in the $i$'th row of a particular column has a "yes" if the task specification corresponding to this column is $i$-feasible. System $(\boldsymbol{u}_k, \boldsymbol{s}_k, \boldsymbol{n}_k, m)$ is thus feasible if and only if the table, when filled, contains a "yes" in the $m$'th row of the column corresponding to the task specification $\boldsymbol{n}_k$. Observe that the table consists of $m$ rows and at most $[(n/k)+1]^k$ columns; and is hence of size polynomial in $n$ for constant $k$. Our algorithm fills this table in a row at a time (beginning with the first row), and takes polynomial time to fill in each cell; consequently, the entire feasibility algorithm is polynomial in the number of task instances $n$.

**Filling in the first row:** Each cell in the first row of the feasibility table is filled with a "yes" or a "no" depending upon whether the task specification corresponding to the cell's column is 1-feasible or not.

Observe that a task specification $\boldsymbol{\nu}_k$ 1-feasible iff

$$\left[\left(\sum_{\ell=1}^{k} u_\ell \cdot \nu_\ell\right) \leq 1\right] \quad \text{and} \quad \left[\left(\sum_{\ell=1}^{k} s_\ell \cdot \nu_\ell\right) \leq 1\right] . \tag{1}$$

Since these conditions can be checked in polynomial time for any task specification, it follows that each cell in the first row is filled in polynomial time.

**Filling in the $i$'th row:** We assume that all the rows $1, 2, \ldots, (i-1)$ have been filled, and describe how the $i$'th row is filled, for any $i$, $1 < i \leq m$. We consider both notions of feasibility that we discussed above – the *replication* based notion (in which each instance of a task on a processor needs its own copy of the code), and the *non-replication* based notion (in which different instances of the same task on a processor share the same code).

- **Replication based:** Task specification $\boldsymbol{\nu}_k$ is $i$-feasible iff there is some task specification $\boldsymbol{\nu'}_k$ that is $(i-1)$-feasible, and

$$\left[\left(\sum_{\ell=1}^{k} u_\ell \cdot \max\{0, \nu_\ell - \nu'_\ell\}\right) \leq 1\right] \quad \text{and} \quad \left[\left(\sum_{\ell=1}^{k} s_\ell \cdot \max\{0, \nu_\ell - \nu'_\ell\}\right) \leq 1\right] \tag{2}$$

  This is because $\nu_\ell - \nu'_\ell$ additional type-$\ell$ tasks must be accommodated on the $i$'th processor if one is to "extend" a partitioning of $\boldsymbol{\nu'}_k$ upon $(i-1)$ processors to a partitioning of $\boldsymbol{\nu}_k$ on $i$ processors, and each such task needs its entire computing capacity and memory, for each $\ell$, $1 \leq \ell \leq k$.
- **Non-replication based:** Task specification $\boldsymbol{\nu}_k$ is $i$-feasible iff there is some task specification $\boldsymbol{\nu'}_k$ that is $(i-1)$-feasible, and

$$\left[\left(\sum_{\ell=1}^{k} u_\ell \cdot \max\{0, \nu_\ell - \nu'_\ell\}\right) \leq 1\right] \text{ and } \left[\left(\sum_{\{\text{all } \ell \mid \nu_\ell > \nu'_\ell\}} s_\ell\right) \leq 1\right] \tag{3}$$

As above, this is because $\nu_\ell - \nu'_\ell$ additional type-$\ell$ tasks must be accommodated on the $i$'th processor in extending a partitioning of $\boldsymbol{\nu'}_k$ upon $(i-1)$ processors to a partitioning of $\boldsymbol{\nu}_k$ on $i$ processors. Each such added task needs its entire computing capacity on the $i$'th processor; however, memory need be reserved for type-$\ell$ tasks only if $\nu_\ell > \nu'_\ell$, and if so only one copy of the type-$\ell$ task code need be stored on this processor.

Under either notion of feasibility, for each cell in the $i$'th row the task specification corresponding to each cell in the $(i-1)$'th row with a "yes" in it is a potential candidate for $\boldsymbol{\nu'}_k$ in the conditions above — since there are polynomially many columns, there are polynomially many such candidates. For a given $\boldsymbol{\nu'}_k$), the conditions can be checked in polynomial time; consequently, each cell in the $i$'th row can be filled in polynomial time.

### 3.2 Obtaining a partitioning

By using standard techniques from dynamic programming, the feasibility analysis algorithm of Section 3.1 above is easily extended to actually obtain a feasible mapping from the set of tasks to the set of processors. In order to do so, one would need to retain some additional information during the filling in of the feasibility table. Specifically, for each cell in the feasibility table that is filled with a "yes," one would need to keep track of the identity of the cell in the *previous* row of the feasibility table corresponding to the mapping that was extended to obtain the partitioning corresponding to the current cell. Once the feasibility of the system has been determined (by having a "yes" in the cell in the last row of the feasibility table that correponds to the entire task set), one could use this information to recursively determine the tasks that go on each processor.

### 3.3 Extensions

The feasibility-analysis and partitioning algorithms described above could be generalized in several ways:

- Although we have focused upon two constrained resources – the execution requirement and the code-size – only, it is relatively straightforward to generalize to a larger number of resources (such as communication bandwidth, energy, etc.) that are available in limited quantities on each processor. Suppose that there are $k$ such constrained resources per processor, and that each task is characterized by its need of each of these $k$ resources. We would need to extend the rules used to fill in the cells of the feasibility table to take all $k$ resources into account; we state without proof that this would be achieved by making Conditions 1, 2, and 3 each be a conjunct of $k$ distinct conditions, one corresponding to each resource type, and that making this modification would retain the polynomial time run-time for the algorithm for filling in the feasibility table.

– We also observe that the algorithm does not require that the processors all be identical – given a multiprocessor platform in which different processors have different computing capacities and different amounts of local memory, the algorithm above is easily modified to be able to perform feasibility analysis and partitioning upon such platforms as well. We state without proof that on such platforms, the order in which the processors are considered — i.e., the mapping of processors to rows of the feasibility table — may effect the run-time of the feasibility-analysis algorithm, but this order would not compromise correctness, nor would it render the algorithm non- polynomial-time. We also assert without proof that the feasibility-analysis algorithm can also be extended to handle heterogeneous multiprocessor platforms – platforms in which different tasks execute at fundamentally different rates, and have fundamentally different memory requirements, upon different processors[2].

## 4 Conclusions

The memory-constrained multiprocessor partitioning problem is a formalization of the problem of mapping periodic real-time tasks to processors in multiprocessor environments in which both computing capacity and local memory are available in limited quantities on the individual processors. We have previously [3] shown that this problem is intractable (NP-hard in the strong sense).

In this paper, we have considered a special case of the memory-constrained multiprocessor partitioning problem, in which all the tasks to be partitioned are separate instantiations of a few distinct types of tasks. We believe that this case is an accurate model for many actual applications (including the application domain of network processors that is of particular interest to us). For this special case, we have obtained exact polynomial-time feasibility tests under two different assumptions regarding how much memory is needed if multiple invocations of the same task co-exist on the same processor. Our feasibility tests are easily extended to determine an actual partitioning of the tasks upon the processors comprising the platform.

In our opinion, one of the most interesting avenues of theoretical research that remain to be explored on this problem concern systems in which the set of tasks to be executed is not static, but *varies dynamically* during the run-time of the systems. Once again, our application domain of network processors motivates interest in this problem, since an NP will typically react to changes in the incoming network traffic by changing the coctail of tasks that is is executing. For such dynamic workloads, the challenge is to devise *suitable metrics* for determining when one partitioning algorithm performs better than another; designing *on-line algorithms* that can react to changes in the workload by efficiently computing a new task-to-processor mapping which does not disrupt the current mapping more than is necessary; and proving properties concerning the *stability* and the *competitiveness* of such dynamic re-mapping algorithms.

[2] For a formal definition of such platforms, see, e.g., [17].

## References


1. Oh, D.I., Baker, T.P.: Utilization bounds for N-processor rate monotone scheduling with static processor assignment. Real-Time Systems: The International Journal of Time-Critical Computing **15** (1998) 183–192
2. Lopez, J.M., Garcia, M., Diaz, J.L., Garcia, D.F.: Worst-case utilization bound for EDF scheduling in real-time multiprocessor systems. In: Proceedings of the EuroMicro Conference on Real-Time Systems, Stockholm, Sweden, IEEE Computer Society Press (2000) 25–34
3. Fisher, N., Anderson, J., Baruah, S.: Task partitioning upon memory-constrained multiprocessors. Submitted for publication (2004)
4. Johnson, D.S.: Near-optimal Bin Packing Algorithms. PhD thesis, Department of Mathematics, Massachusetts Institute of Technology (1973)
5. Johnson, D.: Fast algorithms for bin packing. Journal of Computer and Systems Science **8** (1974) 272–314
6. Garey, M.R., Graham, R.L., Johnson, D.S., Yao, A.C.C.: Resource constrained scheduling as generalized bin packing. Journal of Combinatorial Theory (Series A) **21** (1976) 257–298
7. Woeginger, G.J.: There is no asymptotic PTAS for two-dimensional vector packing. Information Processing Letters **64** (1997) 293–297
8. Caprara, A., Toth, P.: Lower bounds and algorithms for the 2-dimensional vector packing problem. Discrete Applied Mathematics **111** (2001) 231–262
9. de la Vega, W.F., Lueker, G.S.: Bin packing can be solved within $1 + \epsilon$ in linear time. Combinatorica **1** (1981) 349–355
10. Maruyama, K., Tang, D.T., Chang, S.K.: A general packing algorithm for multidimensional resource requirements. International Journal of Computer and Information Sciences **6** (1977) 131–149
11. Yao, A.C.C.: New algorithms for bin packing. Journal of the ACM **27** (1980) 207–227
12. Beck, J., Siewiorek, D.: Modeling multicomputer task allocation as a vector packing problem. In: Proceedings of the 9th International Symposium on System Synthesis, San Deigo, CA (1996) 115–121
13. Madsen, J., Bjorn-Jorgensen, P.: Embedded system synthesis under memory constraints. In: International Workshop on Hardware/Software Co-Design (CODES), Rome, Italy, ACM Press (1999)
14. Szymanek, R.W., Kuchcinski, K.: A constructive algorithm for memory-aware task assignment and scheduling. In: International Workshop on Hardware/Software Co-Design (CODES), Copenhagen, Denmark, ACM Press (2001)
15. Szymanek, R.W., Kuchcinski, K.: Partial task assignment of task graphs under heterogeneous resource constraints. In: International ACM/ IEEE Design Automation Conference (DAC), Anaheim, CA, ACM Press (2003) 244–249
16. Grandpierre, T., Lavarenne, C., Sorel, Y.: Rapid prototyping for real-time embedded heterogeneous multiprocessors. In: International Workshop on Hardware/Software Co-Design (CODES), Rome, Italy, ACM Press (1999)
17. Baruah, S.: Partitioning real-time tasks among heterogeneous multiprocessors. In: Proceedings of the Thirty-third Annual International Conference on Parallel Processing, Montreal, Canada, IEEE Computer Society Press (2004)