Multiprocessor Clustering for Embedded Systems[1]

Vida Kianzad and Shuvra S. Bhattacharyya
University of Maryland at College Park
{vida, ssb}@eng.umd.edu

Abstract. In this paper, we address two key trends in the synthesis of implementations for embedded multiprocessors — (1) the increasing importance of managing interprocessor communication (IPC) in an efficient manner, and (2) the acceptance of significantly longer compilation time by embedded system designers. The former aspect is evident in the increasing interest among embedded system architects in innovative communication architectures, such as those involving optical interconnection technologies, and hybrid electro-optical structures [7]. The latter aspect results because embedded multiprocessor systems are typically designed as final implementations for dedicated functions. While multiprocessor mapping strategies for general-purpose systems are usually designed with low to moderate complexity as a constraint, embedded system design tools are allowed to employ more thorough and time-consuming optimization techniques.

1. Introduction

In this paper, we develop novel partitioning and scheduling techniques that aggressively streamline interprocessor communication. We address the increasing importance of managing interprocessor communication in an efficient manner. This importance is due to the increasing interest among embedded system architects in innovative communication architectures, such as those involving optical interconnection technologies, and hybrid electro-optical structures [7]. Effective experimentation with unconventional architectures requires adequate design tools that can exploit such architectures. We also address the increased compile time tolerance in embedded system design. This results because embedded multiprocessor systems are typically designed as final implementations for *dedicated* functions; modifications to embedded system implementations are rare, and this allows embedded system design tools to employ more thorough, time-consuming optimization techniques. In contrast, multiprocessor mapping strategies for general purpose systems are typically designed with low to moderate complexity as a constraint.

Our work builds on the two-phased decomposition of multiprocessor scheduling that was introduced by Sarkar [5], and explored subsequently by other researchers such as Yang and Gerasoulis [10]. In this decomposition, the application graph is first mapped to a *fully-connected* multiprocessor architecture that has an unbounded number of processors. The goal of mapping here is to minimize the net execution time. In the second phase of Sarkar's two-phase process, called *merging*, the derived schedule is mapped onto the given resource-constrained architecture. Our use of Sarkar's decomposition scheme and the associated breakdown of scheduling into different phases is motivated by the idea of

1.This research was sponsored by the Defense Advanced Research Projects Agency, and the U. S. National Science Foundation.

R. Sakellariou et al. (Eds.): Euro-Par 2001, LNCS 2150, pp. 697-701, 2001.

introducing modularity, and hence more flexibility, in allocating compile-time resources throughout the optimization process. In this paper, we focus on the first phase (*clustering*) of this decomposed problem. Algorithms to address second phase have been discussed in [8].

2. Background and Previous Work

In the context of embedded system implementation, one limitation shared by many scheduling algorithms is that they have been designed for general purpose computation. In the general-purpose domain, there are many applications for which short compile time is of major concern. In such scenarios, it is highly desirable to ensure that an application can be mapped to an architecture within a matter of seconds. Sarkar's internalization algorithm (SIA) and the dominant sequence clustering algorithm (DSC)[10] provide such low complexity algorithms.

However, being deterministic in nature, neither SIA nor DSC can exploit the increased compile time tolerance in embedded system implementation. There has been some probabilistic search implementation of scheduling heuristics in the literature, such as genetic algorithms (GAs), that exploit this increased compile time tolerance. Hou et al. [2], Wang and Korfhage [9], Kwok and Ahmad [4], Zomaya et al. [12], and Correa et al. [1] have proposed different genetic algorithms in the scheduling context. Hou and Correa use similar integer string representations of solutions. Wang and Korfhage use a two-dimensional matrix scheme to encode the solution. Kwok and Ahmad also use integer string representations, and Zomaya et al. use a matrix of integer substrings. All of these algorithms have relatively complex solution representations in the underlying GA formulation. We show that in the context of the clustering/merging decomposition, we can avoid these complications in the clustering phase, and use more streamlined solution encodings for clustering.

We have explored a number of approaches for exploiting the increased compile-time tolerance in embedded multiprocessor implementation, the first of which applies the concept of GAs to develop a novel approach for multiprocessor clustering and scheduling. We represent the applications that are to be mapped into parallel implementations in terms of the widely-used *task graph model*. A task graph is a directed acyclic graph (DAG) $G = (V, E)$, where

- V is the set of task nodes, which are in one-to-one correspondence with the computational tasks in the application ($(V = \{v_1, v_2, \ldots, v_{|V|}\})$).
- E is the set of communication edges (each member is an ordered pair of tasks).
- $t : V \rightarrow \aleph$ denotes a function that assigns an execution time to each member of V.
- $C : V \times V \rightarrow \aleph$ denotes a function that gives the cost (latency) of each communication edge. That is, $C(v, v) \equiv 0$ for all V; $C(v_1, v_2) = C(v_2, v_1)$ for all v_1, v_2; and $C(v_1, v_2)$ is the cost of transferring data between v_1 and v_2 if they are assigned to different processors.

The **net execution time** is defined by the following expression:

$$\tau_N = max(tlevel(v_x) + blevel(v_x) \mid v_x \in V), \quad (1)$$

where $tlevel(v_x)$ ($blevel(v_x)$) is the length of the longest path between node v_x and the source (sink) node in the *scheduled graph*, including all of the communication and computation costs in that path, but excluding $t(v_x)$ from $tlevel(v_x)$. Here, by the scheduled graph, we mean the task graph with all known information about clustering and task execution ordering modeled using additional zero-cost edges. In particular, if v_1 and v_2 are clustered together, and v_2 is scheduled to execute immediately after v_1, then the edge (v_1, v_2) is inserted in the scheduled graph.

3. Solution Representation

We propose a new framework for applying GAs to scheduling problems. Our solution representation encodes scheduling-related information as a single subset of graph edges β, with no notion of an ordering among the elements of β. This representation can be used with a wide variety of scheduling and clustering problems. It exploits the view of a clustering as a subset of edges in the task graph. Gerasoulis and Yang have suggested this view of clustering in their characterization of certain clustering algorithms as being *edge-zeroing* algorithms [10]. One of our contributions in this paper is to apply this subset-based view of clustering to develop a natural, efficient genetic algorithm formulation. For the purpose of a genetic algorithm, the representation of graph clusterings as subsets of edges is attractive since *subsets have natural and efficient mappings into the framework of genetic algorithms.*

Derived from the *schema* theory (a schema denotes a similarity template that represents a subset of $\{0, 1\}^n$), canonical GAs (which use binary representations of solution spaces) provide near-optimal sampling strategies. Furthermore, binary encodings in which the semantic interpretations of different bit positions exhibit high symmetry (e.g., in our case, each bit corresponds to the existence or absence of an edge within a cluster) allow us to leverage extensive prior research on genetic operators for symmetric encodings rather than forcing us to develop specialized, less-thoroughly-tested operators to handle the underlying non symmetric representation. Accordingly, our binary encoding scheme is favored both by schema theory, and significant prior work on genetic operators. Furthermore, by providing no constraints on genetic operators, our encoding scheme preserves the natural behavior of GAs.

Our approach to encoding clustering solutions is based on the following definition.

Definition 1: Suppose that β is a subset of task graph edges. Then $f_\beta : E \rightarrow \{0, 1\}$ denotes the **clusterization function** associated with β. This function is defined by:

$$f(e_i) = \begin{cases} 0 \text{ if } (e_i \in \beta) \\ 1 \text{ otherwise} \end{cases} . \tag{2}$$

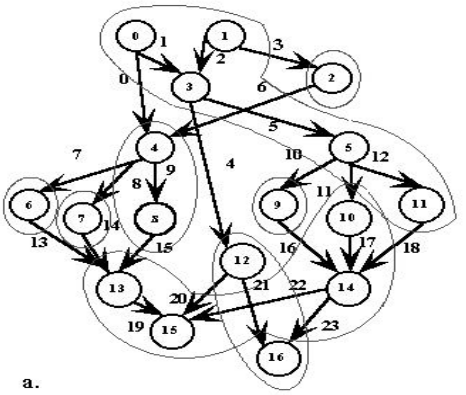
When using a clusterization function to represent a clustering solution, the edge subset β is taken to be the set of edges that are contained in clusters. An illustration is shown in Figure 1. Because it is based on using clusterization functions to represent candidate solutions, we refer to our GA approach as the *clusterization function algorithm* (CFA).

In the CFA, the initial population is initialized with a random selection of clusterization functions (mappings from E into $\{0, 1\}$) and its fitness is evaluated from the net execution time τ_N (from (1)). To compute τ_N, we have applied a modified version of list scheduling that abandons the restrictions imposed by a global scheduling clock, as proposed in [6]. More details on our implementation can be found in [3].

4. Performance Evaluation and Comparison

In this section, we present an experimental comparison of DSC, SIA and CFA. To be fair in comparison of these algorithms (DSC and SIA are deterministic heuristics, while our GA is a guided probabilistic search method), we have implemented randomized versions of DSC and SIA — each such randomized algorithm, like CFA, can exploit increases in additional computational resources to explore larger segments of the solution space.

We have incorporated randomization into to the edge selection process when deriving randomized versions of DSC (RDSC) and SIA (RSIA). In the randomized versions, the



β = {{1,2,5,12},
{9},
{17,19,22},
{21}}

b.

Cluster	Nodes
c1	0,1,3,5,11
c2	2
c3	4,8
c4	6
c5	7
c6	9
c7	10,13,14,15
c8	12,16

e_0	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_{11}	e_{12}	e_{13}	e_{14}	e_{15}	e_{16}	e_{17}	e_{18}	e_{19}	e_{20}	e_{21}	e_{22}	e_{23}
1	0	0	1	1	0	1	1	1	0	1	1	0	1	1	1	1	0	1	0	1	1	0	1

c.

Figure 1. (a) A clustering of an application graph; (b) the corresponding subset β of "zeroed" edges; (c) the corresponding clusterization function f_β.

first element of the sorted edge list — the candidate to be zeroed — is selected with probability p, (we call p the *randomization parameter*); if this element is not chosen, the second element is selected with probability p; and so on, until some element is chosen, or no element is returned after considering all the elements in the list. In this last case (no element is chosen), a random number is chosen from a uniform distribution over $\{0, 1, \ldots, |T| - 1\}$ (where T is the set of edges that have not yet been clustered). Further details on this general approach to incorporating randomization into greedy, priority-based algorithms can be found in [11], which explores randomization techniques in the context of DSP memory management.

Each randomized algorithm begins by first applying the underlying (original) deterministic algorithm, and then repeatedly computing additional solutions with a "degree of randomness" determined by p. The best solution computed within the allotted compile-time tolerance (e.g., 10 min., 1 hr., etc.) is returned. The allotted running time for each input graph to RDSC or RSIA was determined from the CFA running time on the same graphs for 3000 iterations, which allows comparison under equal amounts of running time.

All the heuristics have been tested with two sets of input graphs, DSP-related task graphs and random graphs (with 50 to 1000 nodes). We have also varied the *communication to computation cost ratio* (CCR), between 0.1 to 10 when experimenting with each task graph. The net execution times of random graphs for all algorithms are shown in Figure 2 (more

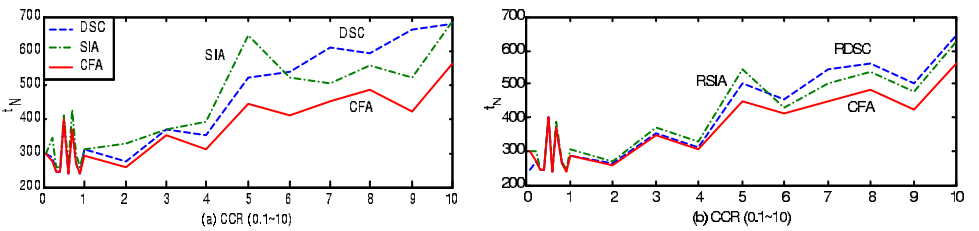


Figure 2. τ_N of different heuristic and randomized algorithms for random graphs.

results, including those on the DSP-related graphs, can be found in [3]). It can be seen from the figure that CFA consistently performs significantly better than the other approaches, and the benefit of the CFA approach increases with increasing CCR values. Overall, for both random and application-related graphs, our experimental results [3] show that CFA is preferable for compile time tolerances that accommodate the underlying GA configuration (less than 1 min. to 10 hrs for the graphs that we considered in our experiments).

5. Summary and Conclusions

This paper has explored multiprocessor clustering techniques to exploit the increased compile time tolerance of the embedded systems domain, and achieve efficient mapping of applications onto multiprocessor architectures. We have developed a novel and natural genetic algorithm formulation, called CFA, for multiprocessor clustering, as well as randomized versions, called RDSC and RSIA, of two well-known deterministic algorithms, DSC [10] and SIA [5], respectively. RDSC and RSIA perform at least as well as DSC and SIA, but are able to exploit arbitrary increases in compile time tolerance due to their incorporation of probabilistic selection. Based on these developments, we have performed an extensive experimental study that compares the alternative strategies under equal amounts of running time (compile time tolerance). Our experiments have demonstrated that the CFA algorithm significantly outperforms RDSC and RSIA, and that the improvement offered by CFA increases with increasing communication costs in the application relative to the amount of computation. Thus, CFA is especially useful when managing communication costs is important. Presently, we are developing further experiments to quantify these distinctions. Another useful direction for further work is exploring the integration of merging algorithms into the CFA framework (e.g., in the fitness evaluation phase).

References

1. R.C. Correa, A. Ferreira, P. Rebreyend, "Scheduling Multiprocessor Tasks with Genetic Algorithms," *IEEE Trans. on Parallel and Distributed Sys.*, Vol. 0, 825-837, 1999.

2. E.S. H. Hou, N. Ansari, H. Ren, "A Genetic Algorithm for Multiprocessor Scheduling," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 5, 113-120, 1994.

3. V. Kianzad, S. S. Bhattacharyya, Multiprocessor clustering for embedded system implementation. Tech. report, Inst. for Advanced Computer Studies, UMCP June 2001.

4. Y. Kwok, I. Ahmad, "Efficient Scheduling of Arbitrary Task Graphs to Multiprocessors Using A Parallel Genetic Algorithm," *Journal of Parallel and Distributed Computing*, 1997.

5. V. Sarkar. *Partitioning and Scheduling Parallel Programs for Multiprocessors*. MIT Press, 1989.

6. G. C. Sih, E. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures." *IEEE Trans. on Parallel and Dist. Sys.*, Vol. 4, No. 2, 1993.

7. D. Spencer, J. Kepner, D. Martinez, "Evaluation of advanced optoelectronic interconnect technology," MIT Lincoln Laboratory August 1999.

8. T.Yang, A. Gerasoulis, "PYRROS: States scheduling and code generation for message passing multiprocessors," *Proc. of 6th ACM Int. Conf. on Supercomputing*, 1992.

9. P. Wang, W. Korfhage, "Process Scheduling Using Genetic Algorithms," *IEEE Symp. on Parallel and Distributed Processing*, 638-641, 1995.

10. T.Yang, A.Gerasoulis, "DSC: scheduling parallel tasks on an unbounded number of processors," *IEEE Trans. on Parallel and Distributed Sys.*, Vol. 5, 951-967, 1994.

11. E. Zitzler, J. Teich, S. S. Bhattacharyya. Optimized software synthesis for DSP using randomization techniques. Tech. report, Swiss Federal Institute of Technology, Zurich, July 1999.

12. A.Y. Zomaya, C. Ward, B. Macey, "Genetic scheduling for parallel processor systems: comparative studies and performance issues," *IEEE Trans. on Parallel and Dist. Sys.*, Vol. 10, 795-812, 1999.