An Extended GCLP algorithm for hardware Software partitioning

Anurag Srivastava

M.Tech (IT) GGS Indraprastha University Delhi
anurag.cs@gmail.com

ABSTRACT
The meeting of system-level objectives by exploiting the trade-offs between hardware and software in a system through their concurrent design. One of the most crucial steps in the design of embedded systems is hardware-software partitioning, i.e. deciding which components of the system should be implemented in hardware and which ones in software. In this paper the extended global criticality/local phase (GCLP) driven algorithm is proposed.

KEYWORDS
GCLP, hardware software codesign, embedded system

INTRODUCTION
The application specific hardware is much faster than software, since hardware runs parallel, but it is also more expensive. Software is cheap and provides the great flexibility to modify, append the features with it along with cheaper to create and maintain. But software is slow with respect to hardware due to its sequential execution nature. Hence in ideal system performance critical components should be realized in hardware and non critical components with variable values to be in software. It is very challenging to create an ideal system. The composition of hardware and software elements also creates new problem, e.g. related to the communication of hardware and software components, as well as other system architecture issues. Therefore partitioning has dramatic impact on the cost and performance of the whole system. Traditionally, partitioning was carried out manually, however, as the system to design have become more and more complex so need of portioning automation arises.

PREVIOUS WORK

The diversity of the requirements has led to extremely heterogeneous system architectures, whereas the short design cycles boosted the demand for early design decisions, such as architecture selection and HW/SW partitioning on the highest abstraction level, i.e. the algorithmic description of the system. HW/SW partitioning can in general be described as the mapping of the interconnected functional objects that constitute the behavioral model of the system onto a chosen architecture model [1]. The task of partitioning has been thoroughly researched and enhanced and produced a number of feasible solutions, which depend heavily on their prerequisites: the architecture model, the communication model, the granularity of the functional objects, etc.

One of the most relevant works are presented in [2]: a very sophisticated integer linear programming model for the joint partitioning and scheduling problem for a wide range of target architectures. This integer program is part of a 2-phase heuristic optimization scheme which aims at gaining better and better timing estimates using the repeated scheduling phases, and using the estimates in the partitioning phases[3]. Given a specific architecture, partitioning of a system level functional description results in a labeling of its tasks as hardware or software operations. The exact solution of such a partitioning problem, even in the simplest cases, requires a solution to computationally intractable problems. In an attempt to mathematically model the variables affecting the partitioning problem, integer programming (IP) and integer linear programming (ILP) formulations have been proposed [4][5]. Comparison of mathematical programming approaches to hardware/software partitioning is difficult, because the quality of results is often strongly affected by the parametric accuracy of the variables used and by the complexity of the cost/performance model[6]. Heuristic approaches to partitioning consist primarily of two strategies: constructive methods such as clustering techniques and iterative methods such as network flow, binary-constraint search, and dynamic programming. By far, the most used methods are based on variable-depth search methods such as variants of the Kernighan–Lin (KL) migration heuristics, or probabilistic hill-climbing methods such as simulated-annealing or genetic algorithms. Heuristic approaches dominate the field of partitioning algorithms, since partitioning is known to be an NP-hard problem in most formulations [7].

FORMAL PROBLEM DEFINITION:
The partitioning problem is to assign n objects O = {o1, ..., on} to m blocks (also called partitions) P= {p1, ..., pm}, such that
- p1 U p2 ... U pm = O
- pi ∩ pj = { } for every i,j : i ≠ j
- cost c(P) are minimized.

In system synthesis
- Objects = data flow graph nodes
- Blocks = architecture graph nodes

Cost Functions:
The component of system is divided in to hardware and software on the basis of cost of components.[8]
The cost may measure on the following basis:
C … system cost in [$]
L … latency in [sec]
P… power consumption in [W]
Example: linear cost function with penalty
f(C, L, P) = k₁·hₐ(C, Cmax) + k₂·hₐ(L, LMAX) + k₃·hₐ(P, PMAX)

hₐC, hₐL, hₐP denote how strongly C, L, P violate the design constraints Cmax, Lmax, Pmax

k₁, k₂, k₃ … weighting and normalization

**GCLP Algorithm**

The task level description of an application is translated into Directed Acyclic Graph (DAG) where each node represents a computation and the arcs represent the data and control precedence between the nodes [9][10]. A hardware/software partitioning problem is to decide the mapping for each node of the DAG, either into software or hardware and arrive at the schedule. This partitioning aims at optimizing the overall design (in terms of area and speed of implementation).

This partitioning problem is computationally intensive. For solving this constrained optimization problem, exact solutions through Integer Linear Programming is intractable. In embedded system design the term partitioning combines two tasks: allocation, i.e. the selection of architectural components, and mapping, i.e. the binding of system functions to these components [11]. Usually a number of requirements, or constraints, are to be met in the final solution, for instance execution time, area, throughput, power consumption, etc. This task is known to be a hard optimization problem, in many formulations even NP-hard [12]. The system functionality is typically abstracted into a graph G = (V, E) representation. Vertices V = {a, .. , f} are depicted which are connected by six edges E = {e₁, .., e₆}. The vertices cover the functional objects of the system, or processes, whereas the edges mirror data transfers between different processes. Every vertex has been annotated with characteristic values, that, in the case of the GCLP algorithm for the binary (SW, HW) partitioning problem, build a quadruple: (process computation time (pctsw) and code size (cs) for SW, process computation time (pcthw) and area in gates (gc) for HW) [13]. The edges are annotated with the number of data samples (bytes) transmitted per invocation of one process. Essentially this algorithm is a greedy approach, which visits every vertex exactly once, and decides where to map it based on two different values: the Global Criticality (GC) measure and the Local Phase (LP) measure.

The GC value is a global look-ahead measure that estimates whether time, code size or area is most critical at the current stage of the algorithm and then decides which of these targets shall be minimized. The LP value is calculated for every single process before the main algorithm starts and is based on intrinsic properties that represent the individual mapping preferences of this process [14].

![The GCLP Algorithm by Dr. Kalavade](image)

**EXTENDED GCLP Algorithm**

The analysis and evaluation of the original algorithm disclosed several possibilities to save computation time and to improve quality. The GCLP algorithm is a heuristic for solving the binary partitioning problem in linear time. The list-scheduling algorithm forms the underlying scheduling framework for the GCLP algorithm. Unlike list scheduling, which either optimizes for the finish time of the node or for the area of the node, the GCLP algorithm adaptively selects an appropriate mapping at each step based on global criticality and local phase.

In this proposed algorithm; the graph is created by the random graph generator. Since graph is generated randomly so probability of high efficient is high.

The algorithm takes various randomly created graphs as the input and creates the map for hardware and software module. These modules are stored in the file with cost information as the output of this. This algorithm suggests top N mapping with minimum cost to user and also suggest the best graph in terms of minimum cost. So the final partition is the optimal for the given system. The execution time will be higher for this algorithm with respect to basic GCLP algorithm but the output is most optimized partition.
Conclusions

We discuss here about the codesign approach and available various algorithms. We extended the existing GCLP algorithm and implemented in the ACS domain within Ptolemy and used for know problem to compare the result with the optimal partitioning. The results are matching with the existing solutions. We hybrid the concept of random inputs and computation of graph.

REFERENCES


