Available online at www.sciencedirect.com
SCIENCE DIRECT
Information Processing Letters 98 (2006) 41–46
Information Processing Letters
www.elsevier.com/locate/ipl

Low-complex dynamic programming algorithm for hardware/software partitioning

Wu Jigang *, Thambipillai Srikanthan

School of Computer Engineering, Nanyang Technological University, Singapore 639798

Received 17 February 2005; received in revised form 1 December 2005; accepted 7 December 2005

Available online 17 January 2006

Communicated by F.Y.L. Chin

Abstract

A low-complex algorithm is proposed for the hardware/software partitioning. The proposed algorithm employs dynamic programming principles while accounting for communication delays. It is shown that the time complexity of the latest algorithm has been reduced from $O(n^2 \cdot \mathcal{A})$ to $O(n \cdot \mathcal{A})$, without increase in space complexity, for n code fragments and hardware area $\mathcal{A}$.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Dynamic programming; Algorithms; Complexity; Hardware/software partitioning

1. Introduction

Most modern electronic systems are composed by both hardware and software. In the design of such mixed hardware/software (Hw/Sw) systems, co-design techniques play more important role. It dominantly affects to overall system performance [1–5]. Hw/Sw partitioning has been proposed over the last decade. It transforms an application specification into communicating hardware and software components of an embedded system that exhibit the desired behavior and satisfy the performance constraints. Software is more flexible and cheaper, but hardware is faster. Thus, efficient techniques for Hw/Sw partitioning can achieve results superior to software-only solution.

Earlier approaches in [6–8] are hardware-oriented. They start with a complete hardware solution and iteratively move parts of the system to the software as long as the performance constraints are fulfilled. On the other hand, [2,9,10] are software-oriented, because they start with a software program moving pieces to hardware to improve speed until the time constraint is satisfied. In these approaches performance satisfiability is not part of the cost function. For this reason, the algorithms can easily be trapped in a local minimum.

Many approaches emphasis the algorithmic aspects, e.g., evolution algorithm [11], integer programming [12, 13], simulated annealing algorithm [2,14] and ant system algorithm [15]. These approaches are applied to different architectures and cost functions to provided sub-optimal solution minimizing the application execution time. It is difficult to name a clear winner because there have been no widely accepted benchmarks. Generally, they require more iterations resulting in longer design cycle times as the partitioning problem is NP-complete [16–18].

Despite many heuristics and approaches above, developing exact algorithms to find an optimal solution

* Corresponding author.
E-mail addresses: asjgwu@ntu.edu.sg (J. Wu), astsrikan@ntu.edu.sg (T. Srikanthan).

0020-0190/$ – see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.ipl.2005.12.008

is still very important. Knudsen and Madsen proposed an algorithm called PACE employed in the LYCOS co-synthesis system for partitioning control data flow graphs (CDFG) into hardware and software parts [19, 20]. PACE is the latest dynamic programming approach. Its time complexity is $O(n^2 \cdot \mathcal{A})$ and the space complexity is $O(n \cdot \mathcal{A})$ for n code fragments and the available hardware area $\mathcal{A}$.

Unlike most of the previous work, in this paper, we take a theoretical approach focusing only on the algorithmic properties of hardware/software partitioning. In particular, we do not aim at partitioning for a given architecture, nor do we present a complete co-design environment. Rather, we restrict ourselves to the problem of deciding, based on given cost values, which components of the system to implement in hardware and which ones in software. Our contribution is reducing the time complexity of PACE from $O(n^2 \cdot \mathcal{A})$ to $O(n \cdot \mathcal{A})$ without increasing the space complexity.

2. Preliminaries

All assumptions in this paper are the same as those given in [19,20]. In detail, an application corresponds to a CDFG which is divided into basic scheduling code fragments/blocks (called blocks in short), that may be moved between hardware and software. The application is modeled as a sequence of blocks $B_1, B_2, \ldots, B_n$. The corresponding hardware area, hardware execution time, software execution time and intercommunication delays for each block are provided in advance by a synthesis system, e.g., LYCOS [20]. Fig. 1, cited from [19,20], shows the computational model for Hw/Sw partitioning, in which hardware blocks and software blocks cannot execute in parallel. It is assumed that the adjacent hardware blocks are able to communicate the read/write variables they have in common directly between them without involving the software side. In Fig. 1, a_i denotes the area penalty of moving block B_i to hardware, s_i denotes the inherent speedup of moving block B_i to hardware, and e_i denotes the extra speedup which is incurred because of blocks being able to communicate directly with each other when they are both placed in hardware. The objective is to find the optimal partition to realize the best possible speedup on a given hardware area $\mathcal{A}$.

Let $\mathcal{A}$ correspond to the knapsack size, and the block B_i correspond to the item i of the knapsack problem for $1 \leqslant i \leqslant n$. This problem can be reduced to the standard 0–1 knapsack problem, one of the NP-complete problems, for the particular case where the communications are ignored. It is clear that the problem which considers communication is more difficult than the one that does not, and thus the hardness of the problem considered in this paper is also NP-hard.

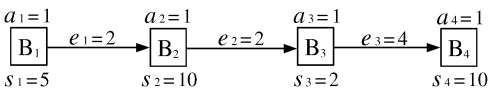

Fig. 1. Computational model of 4 blocks.

3. Algorithms

The algorithm PACE is a dynamic programming approach. It is based on the fact, that *any possible partition can be thought of as composed of sequences of blocks* [19,20], which leads to the higher computational complexity. Let $S_{i,j}$, $j \geqslant i \geqslant 1$, denote the sequence of blocks $B_i, B_{i+1}, \ldots, B_j$. G_j is defined as $\{S_{1,j}, S_{2,j}, \ldots, S_{j,j}\}$, which is called the jth group of the sequence. G_0 is defined as an empty set $\emptyset$. The area penalty $a_{i,j}$ of moving $S_{i,j}$ to hardware is computed as the sum of the individual block areas, i.e., $a_{i,j} = \sum_{k=i}^{j} a_k$. We use following notations to formulize PACE.

1. $speedup(S_{i,j}, a)$ denotes the inherent speedup of moving $S_{i,j}$ to hardware with available area a. For example, in Fig. 1, $speedup(S_{2,3}, 2) = 14$, that is the sum of s_2, e_2 and s_3. While $speedup(S_{2,3}, 1) = 0$ because of not enough hardware area for $S_{2,3}$, i.e., $a_2 + a_3 = 2 > 1$.
2. $Bestsp(G_j, a)$ denotes the best speedup achievable by first moving a sequence from G_j to hardware of area a, and then in the remaining area moving a sequence from one of the previous groups, $G_{j-1}, G_{j-2}, \ldots, G_1$, to hardware. $Bestsp(G_j, a)$ is set to 0 for $G_j = \emptyset$ or $a \leqslant 0$.
3. $Bestsp(G_1 G_2 \cdots G_j, a)$ denotes the best speedup achievable by moving sequences from $G_1, G_2, \ldots$, or G_j to hardware of area a.

The algorithm PACE can be equivalently formulized to (A). The operation max over all values of j returns the maximum of the corresponding set.

$$
\text{(A)}\quad \begin{cases}
Bestsp(G_j, a) = 0 \quad \text{for } j = 0 \text{ or } a \leqslant 0; \\
speedup(S_{i,j}, a) = \begin{cases} 0 & \text{for } a < a_{i,j}; \\ \sum_{k=i}^{j} s_k + \sum_{k=i}^{j-1} e_k & \text{for } a \geqslant a_{i,j}; \end{cases} \\
Bestsp(G_j, a) = \max_{1 \leqslant i \leqslant j} \{ speedup(S_{i,j}, a) + Bestsp(G_{i-1}, a - a_{i,j}) \}; \\
Bestsp(G_1 G_2 \cdots G_j, a) \\
\quad = \max\{ Bestsp(G_j, a), Bestsp(G_1 G_2 \cdots G_{j-1}, a) \}; \\
\quad i \leqslant j, \ j = 1, 2, \ldots, n.
\end{cases}
$$

Download English Version:

https://daneshyari.com/en/article/428437

Download Persian Version:

https://daneshyari.com/article/428437

Daneshyari.com