Piotr DZIURZAŃSKI, Tomasz MĄKA
ZACHODNIOPOMORSKI UNIWERSYTET TECHNOLOGICZNY W SZCZECINIE,
ul. Żołnierska 49, 71-210 Szczecin

Linear optimization of multi-path routing in network on chips

Ph.D. Eng. Piotr DZIURZAŃSKI

He graduated from the Faculty of Computer Science & Information Technology, Szczecin University of Technology in 2000. He obtained the PhD degree at the same Faculty in 2003. Currently he is working at the Chair of Computer Architecture and Telecommunication, West Pomeranian University of Technology. His research interests include hardware/software co-design, high-level synthesis, and formal verification.

e-mail: pdziurzanski@wi.zut.edu.pl

Ph.D. Eng. Tomasz MĄKA

He graduated from the Faculty of Computer Science & Information Technology, Szczecin University of Technology in 2000. He obtained the PhD degree at the same faculty in 2005. Currently he is working at the Chair of Computer Architecture and Telecommunication, West Pomeranian University of Technology. His research interests include audio and speech signal processing and hardware signal processing algorithms' implementations.

e-mail: tmaka@wi.zut.edu.pl

Abstract

In this paper, a technique for determining required link band-width of a multi-path routing algorithm dedicated to Network on Chip (NoC) is presented. The proposed algorithm is based on the linear programming and allows us to avoid deadlocks and contentions in case of Tapeworm routing used for data-dominated streaming multimedia applications realized in Multi Processor Systems on Chip. The proposed approach is illustrated with an example of features extraction module for the Automatic Speech Recognition (ASR) system.

Keywords: Network on Chips, Multi-path routing, features extraction.

Liniowa optymalizacja wielościeżkowego routingu w sieciach wewnątrzukładowych

Streszczenie

W artykule opisano technikę określania wymaganej przepustowości łączy sieci wewnątrzukładowej z routingiem wielościeżkowym. Zaproponowany algorytm bazuje na programowaniu liniowym i umożliwia unikanie blokad w routingu typu Tapeworm, wykorzystywanego dla multimedialnych aplikacji zdominowanych przez dane realizowanych w układach typu MPSoC. Autorski algorytm routingu Tapeworm dla niektórych aplikacji multimedialnych okazuje się być wydajniejszy od XY, powszechnie używanego algorytmu routingu w NoC. Zaproponowane podejście zostało zilustrowane przykładem modułu ekstrakcji cech w systemie automatycznego rozpoznawania mowy. Klasyczny diagram takiego modułu został przedstawiony na rys. 1. W celu określenia marszrut pomiędzy rdzeniami realizującymi funkcjonalności poszczególnych bloków tego modułu została zaadaptowana technika znana z tradycyjnych sieci komputerowych, opisana m.in. w [8]. W artykule zaproponowano sposób wyboru ścieżek między rdzeniem źródłowym i docelowym, opisano sposób określania ograniczeń, a także zaproponowano funkcję celu uwzględniającą długość ścieżki. Do wyszukiwania optymalnej przepustowości łączy wykorzystano algorytm przypominający wyszukiwanie binarne. Badania eksperymentalne, w ramach których zaimplementowano opisany moduł w języku SystemC, a także wykorzystano komercyjne narzędzie do rozwiązywania problemu programowania liniowego, potwierdzają skuteczność i efektywność opisywanego podejścia.

Słowa kluczowe: sieci wewnątrzukładowe, routing wielościeżkowy, ekstrakcja cech.

1. Introduction

A typical multimedia data processing system is usually computational-intensive and dominated by data. It can be split into stages realizable by separate computational units which may benefit from parallel processing in pipelines. Such kind of hardware implementation can be viewed as a kind of a Multi Processor System on Chips (MPSoCs), where each processing unit of a MPSoC realizes a single stage of streaming application processing.

As the amount of data transmitted between these cores can be relatively large, it is essential to guarantee sufficient bandwidth of the connections. The simplest point to point connections require too much space, whereas bus-based connections result in large number of conflicts and, consequently, despite various arbitrage techniques decrease the overall performance of the whole system. Moreover, both these realizations do not scale well with the constantly increasing number of computational units required by contemporary devices dealing with a number of various algorithms in a single system. In order to omit these obstacles, the packet-based Network-on-Chip (NoC) paradigm for designs of chips realizing distributed computation has been introduced.

A classic implementation of a NoC system uses packet switching approach known as wormhole routing [2], where each packet is split into smaller units of equal length, flits (flow control digits). The first flits often contain some information required for the routing process, for example address of the destination node. After obtaining this information, a wormhole router selects the next-hop router and establishes the path to that neighbouring router. This path is utilized exclusively for transferring the considered package flit by flit. If another package is to be sent through the connection already used for transferring a package, its transfer is deferred as long as all flits of the previous package has not been sent. This situation is known as contention and may result in significant decreasing of efficiency. Contentions are especially likely to be observed in various data-intensive applications, where large streams of data are transferred in every second. This may result in violating the real-time constraints and thus making an MPSoC designing in this way inappropriate for the task. The most popular routing algorithm used in NoCs, named XY, can be also viewed as inappropriate for switching large streams of information. According to this algorithm, a flit is firstly routed according to the X axis as long as the X coordinate is not equal to the X coordinate of the destination core, and then the flit is routed vertically. Despite being deadlock-free [7], this algorithm is not adaptive and thus is not equipped with mechanism for decreasing the contention level.

Moreover, as it was shown in [4], in a mesh-based NoC realizing a typical streaming multimedia algorithm, few links are used significantly while the remaining ones are utilized in a small degree and relatively large part of links are not utilized at all.

In our earlier papers, e.g. [4, 3], a multi-path NoC routing algorithm, named Tapeworm, has been proposed. It seems quite promising to data-intensive multimedia algorithms. Basically, this algorithm is based on the well-known Ford-Fulkerson technique and benefits from the fact that in the case of considered MPSoCs we know the core functionalities at the design stage and thus can determine a favorable routing statically. Moreover, we can split a single data stream into a series of sub-streams and route them using different paths.

This technique results in better transfer balance between NoC's paths and requires often lower capacities than the state-of-the-art XY routing protocol. However, we have to compute somehow the capacity of the paths in order to guarantee flawless and deadlock-free routing.

In this paper we propose a technique for the required capacity computation and illustrate it with the example of features extraction for Automatic Speech Recognition (ASR). Our motivation in choosing this kind of application is the fact that speech technologies become more and more prevalent and they require efficient algorithms in its back end. This demand is required for fulfill speech processing time constrains since human speech communication is characterized by its dynamic behaviour.

2. Features extraction for ASR

In the simplest form the Automatic Speech Recognition (ASR) system requires two modules to accomplish speech recognition. The first stage, named feature extraction, performs descriptor values extraction for input speech data, and in the second stage data classification is executed [5]. Data processing efficiency relies on algorithms exploited in both modules and properties of target platform. Depending on the ASR type (isolated, continuous), size of the dictionary, influence of speaker, etc. requirements of the computational and memory resources may vary over a wide range. Therefore, selection of an appropriate software/hardware architecture is crucial for the final ASR processing efficacy [6].

The size of data processed in this stage depends on the properties of input signal (signal length W[ms] and sampling frequency f_s [Hz]). The whole speech signal is divided into N frames, each containing M samples. The frame size (R_s [ms]) and frame overlap (R_o [%]) have the direct impact on the number of frames $N = \lfloor [100 \cdot (W - R_s)]/[R_s \cdot (100 - R_o)] \rfloor$ and their size $M = R_s \cdot f_s/1000$.

Block diagram of extraction process in ASR tasks utilizing the most popular features set is shown in Fig. 1. Consecutive speech frames are fed to input and for each frame a 39-th dimensional vector is calculated. In the result of speech parametrization process $39N$ values representing static and dynamic features are obtained.

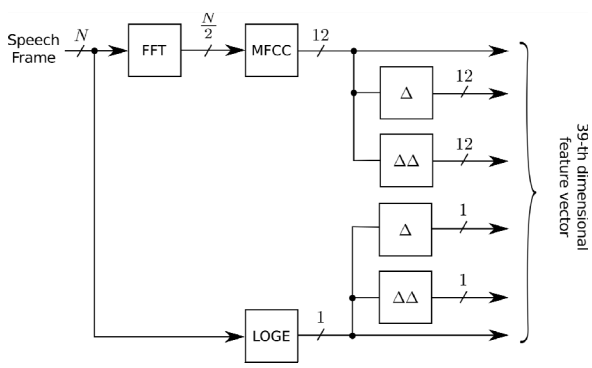

Fig. 1. Block diagram of the classical ASR feature extraction process
Rys. 1. Diagram blokowy klasycznego procesu ekstrakcji cech dla ASR

The logarithm of energy (LOGE) and \$12\$ mel-frequency cepstral coefficients (MFCC) belongs to first type of features. Logarithm of energy is directly calculated from input frame while the MFCC vector is obtained using filter bank in frequency domain (with wrapped frequencies onto the mel scale). During the conversion from time to frequency domain, the output vector of FFT has the half-size of input, because the input data is real-valued.

Since static features cannot handle dynamic nature of speech, additional dynamic features are included: first and second derivatives of LOGE and MFCC [1]. In contrast to static, dynamic features require previously calculated values to determine whole set of descriptors. Therefore, an additional memory is needed to prepare final feature vector.

3. The proposed routing scheme

In order to determine the routing between the cores described in the previous sections, we decided to adapt the technique known from typical computer network that is based on the linear programming approach. We use so-called link-path formulation, whose usage was well explained for computer networks in, e.g., [8]. We will refer to each NoC router as to *node*, as to traffic volume as to *demand volume*. The demand volume is always considered between a pair of nodes, named *demand pair* or *demand*. In our case the demand is always uni-directional, in contrast to the typical computer networks, where it is usually assumed to be bi-directional. We will denote the demand volume as $\hat{h}_{xy}$, where x and y are the source and the destination nodes, respectively.

As it can be seen in Fig. 2a, the demand between two nodes, e.g. 1 and 8, $\langle 1,8 \rangle$, can be routed over a few paths, for example built from the nodes 1,2,5,8 or 1,4,7,8. As it was written in the previous sections, in the tapeworm routing we can split the demand volume between a number of paths. Here we will use $\hat{x}$ with subscripts denoting the nodes along the path to denote the unknown *demand path-flow variable*. For path 1,2,5,8 it is $\hat{x}_{1258}$ and for 1,4,7,8 - $\hat{x}_{1478}$. It is obvious that the demand volume between nodes 1 and 8 we can route using even more paths, and thus we get $\hat{h}_{18} = \hat{x}_{1258} + \hat{x}_{1478} + \hat{x}_{1458} + \hat{x}_{125478} + \hat{x}_{1258} + \hat{x}_{12365478} + \cdots$. In practice we prune this path set as it is described later. Notice that demand path-flow variables cannot be negative for all paths.

The last notation we have to introduce is the *link capacity* (also called *link bandwidth*). Notice that links, in contrast to demand volumes, connects two nodes directly and thus can be considered only for neighboring nodes. We denote links with dash between nodes and the link capacity of the link 1-2 is denoted $\hat{c}_{12}$.

Let us return to the algorithm considered in the previous sections. There we have given that between nodes 1 and 2 there is demand volume equal to $N/2$, so $\hat{h}_{12} = N/2$. Having used the core mapping given in Fig. 2b we can see the following paths between these nodes: 1,2; 1,8,3,2; 1,9,4,2; 1,5,7,8,3,2; 1,5,7,8,1,2; 1,5,6,9,1,2; 1,5,6,9,4,2. However, we can conclude that usage of path 1,5,6,9,1,2 is counter effective, as it includes the direct link 1-2 and a number of other links. So, it is definitely more advisable to use the path 1,2 instead and thus not insert the unnecessary traffic into the remaining links of this path. Thus we can remove this path from our further consideration. Similar reasoning we can perform with path 1,5,6,9,1,2. There exist, obviously, much more paths between nodes 1 and 2, but they also include paths that has been already taken into consideration and thus their usage is not necessary. Concluding, we can write the following equation: $\hat{x}_{12} + \hat{x}_{1832} + \hat{x}_{1942} + \hat{x}_{157832} + \hat{x}_{156942} = \hat{h}_{12} = N/2$.

Without further comments we provide equations for the remaining demand volumes.

$\hat{x}_{15} + \hat{x}_{1875} + \hat{x}_{1965} + \hat{x}_{123875} + \hat{x}_{124965} = \hat{h}_{15} = N$, $\hat{x}_{23} + \hat{x}_{2183} + \hat{x}_{249183} + \hat{x}_{24915783} + \hat{x}_{24965783} + \hat{x}_{215783} = \hat{h}_{23} = 12$, $\hat{x}_{24} + \hat{x}_{2194} + \hat{x}_{215694} + \hat{x}_{21875694} + \hat{x}_{238194} + \hat{x}_{21875194} = \hat{h}_{24} = 12$, $\hat{x}_{56} + \hat{x}_{578196} + \hat{x}_{57832496} + \hat{x}_{57812496} + \hat{x}_{5196} + \hat{x}_{512496} = \hat{h}_{56} = 1$, $\hat{x}_{57} + \hat{x}_{5187} + \hat{x}_{512387} + \hat{x}_{569187} + \hat{x}_{569187} + \hat{x}_{56942187} + \hat{x}_{56942387} = \hat{h}_{57} = 1$.

In our mesh NoC we assume all links to have equal capacity, so $\hat{c}_{12} = \hat{c}_{15} = \cdots = \hat{c}_{96} = \hat{c}$. Our goal is to determine the minimum of link capacity that is possible for the presented algorithm. Then we have to write some constraints. The traffic on each link cannot be higher than its capacity. We have then to limit all demand path-flow variables that include a common link so that the sum of their values is lower or equal to the link capacity.

For example, for link 1-2 we have then $\hat{x}_{12} + \hat{x}_{123875} + \hat{x}_{124965} + \hat{x}_{57812496} + \hat{x}_{512496} + \hat{x}_{512387} \leq \hat{c}_{12}$. Obviously, we have to write such inequalities for each link present in the NoC.

The last equation we have to write is the objective function that we intend to minimize. In this function we plan to penalize longer paths and favor usage of direct links. Thus we decided to multiply each demand path-flow variable by an appropriate weight whose value depends on the path length (in terms of the number of nodes involved). Notice, that in our case the path lengths range from 2 to 8. Denoting the length of each path as i, we introduce weights n_i of the following values: $n_2 = 7$, $n_3 = 6$,..., $n_8 = 1$.

The objective function can be then written in the following way: $F = n_2\hat{x}_{12} + n_4\hat{x}_{1832} + n_6\hat{x}_{157832} + n_6\hat{x}_{159642} + \cdots + n_8\hat{x}_{56942387}$.

In order to determine link capacity $\hat{c}$ of each link (as they are equal each other according to our architecture assumption) we execute the following simple algorithm. We initialize $\hat{c}$ with the value equal to half the sum of all volume demands. Then we execute the linear programming scheme described above. If we find a feasible solution, we divide the $\hat{c}$ by 2; otherwise $\hat{c}$ is increased by $\hat{c}$/2 and the linear programming scheme is executed once again etc. Thus we perform a typical binary search of the optimal $\hat{c}$ value.

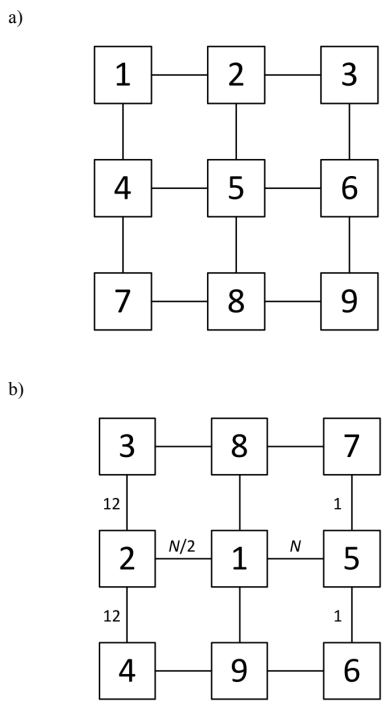

Fig. 2. Cores in the mesh NoC (a), NoC after the core mapping stage (b)
Rys. 2. Rdzenie siatki NoC (a), NoC po etapie odwzorowania rdzeni (b)

4. Implementation and results

The described above ASR blocks and the router architecture has been developed and implemented in CoCentric System Studio (Synopsys and the Synopsys product names described herein are trademarks of Synopsys, Inc.) - a design and simulation environment allowing us to use the SystemC language. It is a C++ library aiming at designing of digital circuits that was developed in order to facilitate production of increasingly more complex systems consisting of a number of components, including the software ones. Thus it is particularly convenient in the case of such complex systems as NoCs.

In order to compute the necessary link bandwidth and the routing paths, we used the technique described in this paper. We have found, solving the appropriate linear programming problem, that the optimal path capacity $\hat{c}$ for $N = 2048$ is equal to 768 flits per frame, which gives 24000 bytes per second. Similarly, for other analysed frame sizes (8ms - 128 ms) we obtained proportional values: the quotient between flits per frame and frame length equals 6. To obtain this result we had to describe the problem in the form of 12 linear equations with 12 variables and constraints in the form of 24 linear inequalities. The value of $\hat{c}$ has been found in the 4th iteration for the binary-search-like algorithm described earlier. It means that if we guarantee such bandwidth of the NoC paths, there should be no contention nor deadlocks and the system should work flawlessly under the real-time constraints.

The total computation can be given in tens of seconds using a typical PC machine with a popular linear program solver.

5. Conclusions

In this paper we have presented an approach for determining the required Network on Chip bandwidth using the proposed multi-path routing algorithm. Our technique utilizes typical linear programming solving methods, adapted from the traditional computer networks. We have shown the applicability of the proposed technique and its result using an example of features extraction system for automatic speech recognition.

Synopsys and the Synopsys product names described herein are trademarks of Synopsys, Inc.

The research work presented in this paper was sponsored by Polish National Science Centre (years 2011-2013).

6. References

[1] Anusuya M., Katti S.: Front end analysis of speech recognition: a review, International Journal of Speech Technology, vol. 14, pp. 99-145, December 2010.

[2] Bjerregaard T., Mahadevan S.: A Survey of Research and Practices of Network-on- Chip, ACM Computing Surveys (CSUR), vol. 38, 2006, Article 1.

[3] Chojnacki B., Maka T., Dziurzanski P.: Virtual path implementation of multi-stream routing in network on chip, 11th international conference on Parallel computing technologies (PaCT'11), LNCS vol. 6783, pp. 431-436, 2011.

[4] Dziurzanski P., Maka T.: Stream Transfer Balancing Scheme Utilizing Multi-Path Routing in Networks on Chip, 4th International Workshop ARC 2008, 26-28 March, London, UK, ss. 294-299, 2008.

[5] Gold B., Morgan N.: Speech and Audio Signal Processing: Processing and Perception of Speech and Music, John Wiley \& Sons, Inc., 2000.

[6] Jurafsky D. and Martin J. H.: Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, Pearson Education Ltd., London, 2009.

[7] Li M., Zeng Q.A., Jone W.B.: DyXY: a proximity congestion-aware deadlock-free dynamic routing method for network on chip. 43rd ACM IEEE Design Automation Conference (DAC), 2006, pp. 849-852.

[8] Pioro M., Medhi D.: Routing, Flow, and Capacity Design in Communication and Computer Networks, Morgan Kaufmann, 2004.

otrzymano / received: 02.04.2012
przyjęto do druku / accepted: 01.06.2012 *artykuł recenzowany / revised paper*