SVR-NoC: A Performance Analysis Tool for Network-on-Chips Using Learning-based Support Vector Regression Model

Zhiliang Qian[1], Da-Cheng Juan[2,3], Paul Bogdan[2,4], Chi-Ying Tsui[1], Diana Marculescu[2] and Radu Marculescu[2]
[1]Electronic and Computer Enineering, Hong Kong University of Science and Technology, Hong Kong
[2]Eletrical and Computer Engineering, Carnegie Mellon University, Pittsburgh
[3]Machine Learning Department, Carnegie Mellon University, Pittsburgh
[4]Department of Electrical Engineering, University of Southern California, Los Angeles

***Abstract*—In this work, we propose SVR-NoC, a learning-based support vector regression (SVR) model for evaluating Network-on-Chip (NoC) latency performance. Different from the state-of-the-art NoC analytical model, which uses classical queuing theory to directly compute the average channel waiting time, the proposed SVR-NoC model performs NoC latency analysis based on learning the typical training data. More specifically, we develop a systematic machine-learning framework that uses the kernel-based support vector regression method to predict the channel average waiting time and the traffic flow latency. Experimental results show that SVR-NoC can predict the average packet latency accurately while achieving about 120X speed-up over simulation-based evaluation methods.**

***Index Terms*—Network-on-Chip, learning, performance model**

I. Introduction and Related Work

Network-on-Chips (NoCs) have been proposed as a promising solution to solve the complex on-chip communication problems [1]. To obtain the optimal architecture, automatic synthesis with design space exploration is required. Because of this, NoC latency models are widely adopted instead of the time consuming simulations to allow fast evaluations, so to explore a larger design space [2].

Most NoC latency models are based on the classical queuing theory and treat each input channel in the router as an M/M/1, M/G/1/N [2] or G/G/1 [3] queue. Indeed, these models provide good estimations when the following assumptions hold: 1) The packet length satisfies an exponential distribution, and therefore the packet service time in the router is exponentially distributed as well [2], [3]. 2) The traffic is assumed to follow a Poisson distribution at all traffic sources [2], [4].

However, in real applications, these assumptions may not hold and the accuracy of the analytical model is compromised. In order to relax some of the assumptions above, in this work, we develop machine-learning techniques to model the NoC latency. Given the target NoC platform, we propose a learning-based method to create an estimation model by training with data-sets obtained from the latency profiles. *Support vector regression* (SVR) [5] and *cross validation* (CV) are utilized during the training process to achieve higher accuracy, while avoiding training data overfitting.

978-3-9815370-0-0/DATE13/©2013 EDAA

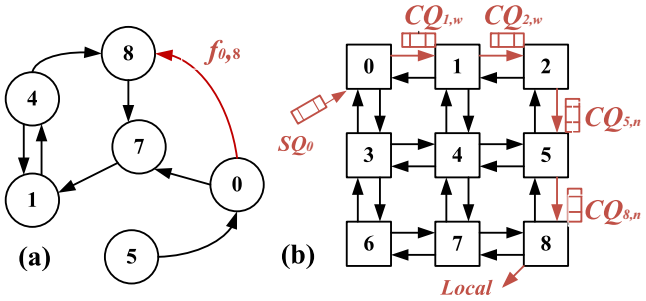

Figure 1. a) An example of core communication graph (CCG) b) The source queue and channel queues in the path from node 0 to 8

The rest of this paper is organized as follows. In Section II, we introduce the basic assumptions, notations and the router model. Section III details the SVR-NoC framework. In Section IV, we present the experimental results. Finally, Section V concludes this work.

II. NoC Router Regression Model

A. Assumptions and basic notations

We target a mesh-based NoC platform with wormhole flow control and deterministic XY routing. Various buffer sizes and pipeline depths are accommodated in the model. As shown in Figure 1, the communication traffic has been characterized by a core communication graph ($CCG(V, F)$), which is a directed graph where each vertex in V denotes a node in the NoC and each edge $f(s, d)$ represents a traffic flow transmitted from the source node s to the destination node d. Table 1 summarizes the parameters and the notations used in the framework.

B. Channel and source queuing regression model

For a specific traffic flow $f_{s,d}$, the average latency between the source s and destination d, $L_{s,d}$, is given by [4]:

$$L_{s,d} = SQ_s + \sum_{(k,dir)\in P_{s,d}}(CQ_{k,dir} + H_s) + \left\lceil \frac{s}{W} \right\rceil \quad (1)$$

The first two terms represent the header flit latency and the third item is the transfer time of its body flits. The overall NoC average latency is then given by:

$$L = \frac{1}{\sum_{s\in V}\sum_{d\in V}\lambda_{s,d}} \sum_{s\in V}\sum_{d\in V}(\lambda_{s,d} \times L_{s,d}) \quad (2)$$

Table 1. Model parameters and definitions

Parameters	Description
W	Input channel width; flit width (bit)
SQ_s	Source queuing time at node s ($\widehat{SQ}_s$: regressed value of SQ_s)
$CQ_{k,dir}$	Input channel queuing time at node k, direction dir ($\widehat{CQ}_{k,dir}$: regressed value of $CQ_{k,dir}$)
S	Packet size (bit)
H_s	Number of router pipeline stages
AR_k	Traffic arrival rate matrix at node k
$AR_{k,dir}$	Traffic arrival rate at node k, direction dir
F_k	Forwarding probability matrix at node k
$F_{k,i,j}$	Packet forwarding probability from input direction i to output direction j at node k
$f_{s,d}$	Application flow from source node s to destination node d
Λ_s	Traffic rate at source node s (packets per cycle)
$\lambda_{s,d}$	Traffic rate from source s to destination d
$L_{s,d}$	Average packet latency from source s to destination d
$P_{s,d}$	Set of node and direction pairs that form the routing path from source s to destination d

As shown in Eq. 1, the source queuing delay SQ_s and average channel waiting time $CQ_{k,dir}$ are the two key components that determine $L_{s,d}$ and L, and so we model them via two regression functions f_{SQ} and f_{CQ}. We denote the features in learning f_{SQ} and f_{CQ} by two vectors $X_{CQ} = [x_{cq1}, x_{cq2}, \cdots, x_{cqn}]$ and $X_{SQ} = [x_{sq1}, x_{sq2}, \cdots, x_{sqn}]$, respectively. The proposed supervised SVR-NoC learning framework is applied on the training data set to formulate the model as the relationship between the selected features and the queuing delays:

$$\widehat{CQ}_{k,dir} = f_{CQ}(X_{CQ}); \qquad \widehat{SQ}_s = f_{SQ}(X_{SQ}) \qquad (3)$$

III. SVR-NoC learning framework

A. SVR-NoC methodology

Figure 2 shows the SVR-NoC framework, which consists of two parts, namely the *training stage* and *prediction engine*. In the training stage, different traffic patterns are fed into the system level NoC simulator to observe the status of the channels in the routers and the source queues. The channel queue feature X_{CQ} includes the packet arrival rates AR and the forwarding probabilities F in the router. The source queue feature X_{SQ} includes the injection rates λ at the traffic source and the neighboring channel waiting times CQ. SVR approach is then used to obtain the f_{CQ} and f_{SQ} models, respectively.

After the training stage, the obtained SVR models are used in the prediction engine to estimate the latency performance for different traffic patterns. The prediction engine can be embedded in the NoC synthesis framework for the inner-loop design evaluation. We first apply the channel regression model f_{CQ} to predict the average channel waiting time $\widehat{CQ}$. Then the predicted $\widehat{CQ}$ features, together with other features extracted from CCG, are applied to the f_{SQ} function to evaluate the source queuing delay. The traffic flow latency and the overall latency are computed according to Equations (1) and (2).

B. Feature extraction for training data

1) Channel queuing feature vector: The average waiting time in the channel dir of router k ($CQ_{k,dir}$) depends not only on its packet arrival rate $AR_{k,dir}$ but also on the contention among the channels within the same router, as well as the traffic conditions in the neighboring routers (such as congestion in the downstream router). We use the arrival rate matrices of the current and the four neighboring routers (N,E,S,W neighbors) together with the forwarding probability matrix F_k in the current router to form the feature vector X_{CQ}. Both arrival rate matrix AR_k and the forwarding probability matrix F_k at node k can be calculated offline based on the application CCG. $AR_{k,dir}$ is given by

$$AR_{k,dir} = \sum_{\forall s \in V} \sum_{\forall d \in V} (\lambda_{s,d} \times R(s,d,k,dir)) \qquad (4)$$

where $R(s,d,k,dir)$ indicates whether the channel is part of the routing path set $P_{s,d}$.

The forwarding probability $F_{k,i,j}$ denotes the portion of traffic that arrives at channel i of node k being forwarded to the output direction j and is given by:

$$F_{k,i,j} = \frac{\sum_{\forall s \in V} \sum_{\forall d \in V} (\lambda_{s,d} \times g(s,d,k,i,j))}{AR_{k,i}} \qquad (5)$$

where $g(s,d,k,i,j)$ returns 1 if the routing path $P_{s,d}$ contains the channel (k,i) through the output direction j, or 0 otherwise.

In Table 2, we list all the elements in the channel queuing feature vector X_{CQ}.

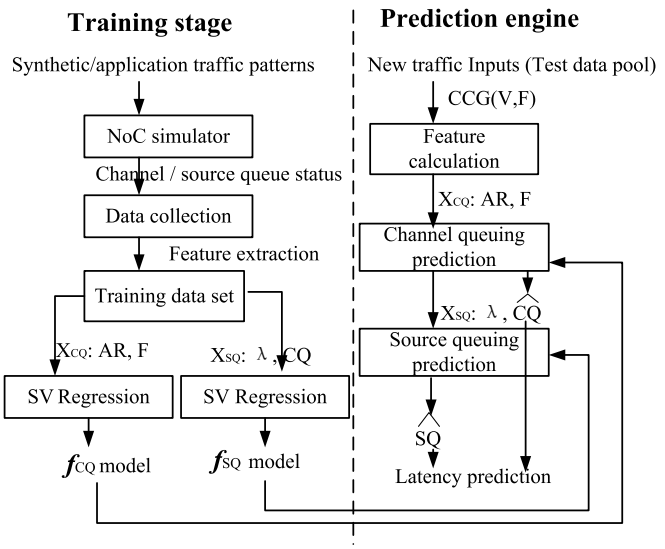

Figure 2. SVR-NoC methodology overview

Table 2. Training features for regression

Feature vector	Elements notation	Elements	Description
X_{CQ}	dir	$X_{CQ}[0]$	The direction of current channel
	AR_k^T	$X_{CQ}[1:5]$	The arrival rate vector of current router k
	$AR_{n(k)}^T$	$X_{CQ}[6:25]$	The arrival rate vector of four neighboring router of k
	F_k	$X_{CQ}[26:50]$	Forwarding probability matrix of router k
X_{SQ}	Λ_s	$X_{SQ}[0]$	Arrival rate at source s
	$CQ_{s,local}$	$X_{SQ}[1]$	Average waiting time in local buffer of router s
	$CQ_{n(s),ch}$	$X_{SQ}[2:5]$	Average waiting time in ch of neighboring router $n(s)$

2) Source queuing feature vector: The time that a packet needs to wait in a particular source queue depends on the traffic generation rate at the processor and the network congestion status, which are reflected by the arrival rate and service rate, respectively. Therefore, the packet generation rate Λ_s and the average waiting time $CQ_{s,local}$ of the local buffer in the router s are first included in X_{SQ} as shown in Table 2.

However, the channel average waiting time $CQ_{s,local}$ cannot be obtained from the input CCG. In the training stage, we extract the $CQ_{s,local}$ from the simulation results. In the prediction engine, we use the estimated $\widehat{CQ}_{s,loal}$ value obtained from the previous f_{CQ} module instead.

As shown in Table 2, to achieve a better inference on the level of network congestion and a higher accuracy, we also need to include the average waiting time of the local router channel $CQ_{n(s),ch}$ in the X_{SQ} vector.

C. Support vector regression for f_{CQ} and f_{SQ}

After obtaining the feature vectors X_{CQ} and X_{SQ}, we apply ϵ-SVR [5] to learn the two nonlinear models f_{CQ} and f_{SQ}, respectively. We will present the ϵ-SVR formulation for the channel average waiting time function f_{CQ} while similar procedures can be applied for f_{SQ}.

1) Primal form formulation: Without loss of generality, we begin by assuming f_{CQ} is a linear function. Given a set of l training data points, *i.e.,* $\{(X_{CQ1}, CQ_1), \cdots (X_{CQl}, CQ_l)\}$, where $X_{CQi} \in \mathbb{R}^d$ is the i^{th} sample feature vector with dimensionality of d ($d = 51$ as shown in Table 2) . Under the linear model assumption, $\widehat{CQ}$ can be expressed as:

$$\widehat{CQ} = f(X_{CQ}) = \sum_{i=1}^{d} w_i x_{cqi} + b = \mathbf{w^T} X_{\mathrm{CQ}} + b \quad (6)$$

where $\mathbf{w} \in \mathbb{R}^d$, $b \in \mathbb{R}$. Let ϵ be the maximum prediction error that we can tolerate. We introduce two slack variables, ξ and ξ^*, to represent the error larger and smaller than the target value by more than ϵ, respectively. In order to prevent data over-fitting when learning $f(X_{CQ})$ from the training samples, a regularization term proportional to $||\mathbf{w}||^2$ is added into the objective function [6]. To find the optimal f_{CQ}, the problem can be formulated as :

$$minimize \quad \tfrac{1}{2}||\mathbf{w}||^2 + C \times \sum_{i=1}^{l}(\xi_i + \xi_i^*) \quad (7)$$

$$subject\ to \begin{cases} (\mathbf{w}^T X_{CQi} + b) - CQ_i \le \epsilon + \xi_i \\ CQ_i - (\mathbf{w}^T X_{CQi} + b) \le \epsilon + \xi_i^* \\ \xi_i, \xi_i^* \ge 0, \epsilon \ge 0 \end{cases} \quad (8)$$

where $C > 0$ is the regularization parameter that determines the tradeoff between the importance of the fitting accuracy and the regularization term for preventing data over-fitting.

2) Dual problem expansion: To solve the optimization problem in (7) and (8), we denote by $\mathcal{L}$ the Lagrangian of the primal form and introduce the Lagrangian multiplies α_i, α_i^*,η_i,η_i^* [6], where:

$$\begin{aligned} \mathcal{L} = & \tfrac{1}{2}||\mathbf{w}||^2 + C \times \sum_{i=1}^{l}(\xi_i + \xi_i^*) - \sum_{i=1}^{l}(\eta_i \xi_i + \eta_i^* \xi_i^*) \\ & - \sum_{i=1}^{l} \alpha_i \times (\epsilon + \xi_i - CQ_i + \mathbf{w}^T X_{CQi} + b) \\ & - \sum_{i=1}^{l} \alpha_i^* \times (\epsilon + \xi_i^* + CQ_i - \mathbf{w}^T X_{CQi} - b) \quad (9) \end{aligned}$$

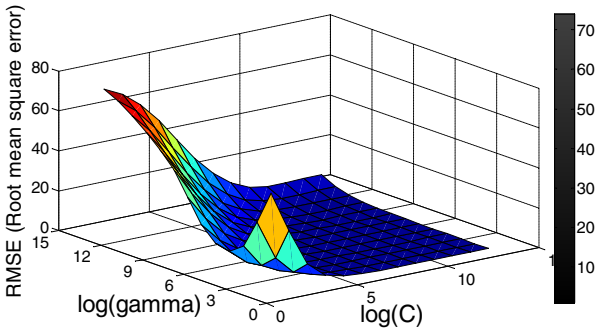

Figure 3. RMSE surface in the grid search

It follows that the partial derivatives of $\mathcal{L}$ with respect to the primal variables $(\mathbf{w}, b, \xi_i, \xi_i^*)$ are zero for the optimality point in the primal form [6]. Hence, the primal variables $(\mathbf{w}, b, \xi_i, \xi_i^*)$ can be represented by α_i, α_i^* after setting $\partial_b \mathcal{L} = \partial_{\mathbf{w}} \mathcal{L} = \partial_{\xi_i} \mathcal{L} = \partial_{\xi_i^*} \mathcal{L} = 0$. By substituting the primal variables in (7) and (8) with their dual variable representations, the dual optimization problem can be obtained [5] as follows:

$$max \quad \begin{cases} -\frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) X_{CQi}^T X_{CQj} \\ -\epsilon \sum_{i=1}^{l}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{l} CQ_i(\alpha_i - \alpha_i^*) \end{cases} \quad (10)$$

$$subject\ to\ \sum_{i=1}^{l}(\alpha_i - \alpha_i^*) = 0\ and\ \alpha_i, \alpha_i^* \in [0, C] \quad (11)$$

This is a quadratic programming problem and can be solved efficiently in polynomial time.

3) Kernel trick for nonlinear extension: In [6], it is shown that the linear model formulation in (10)-(11) is still valid if we substitute $X_{CQi}^T X_{CQj}$ with a variety of kernel functions:

$$k(X_{CQ}, X_{CQ}^{'}) = \Phi(X_{CQ})^T \Phi(X_{CQ}^{'}) \quad (12)$$

From the analytical delay model, it is suggested that the router delay is a non-linear function of the extracted features. Therefore we use the Radial Basis Function (RBF) kernel [6] ($k(\mathbf{x}, \mathbf{x}^{'}) = exp(-\gamma \cdot ||\mathbf{x} - \mathbf{x}^{'}||^2)$), where γ is a tuning parameter to make a non-linear extension of the (10)-(11). The formulation after extension is still a quadratic programming with only a few additonal efforts needed.

D. Cross-validation and grid-search

To find the global optimal learning model, we need to search over different combinations of (ϵ, C, γ) in (7)-(8). In this work, we adopt a v-fold cross-validation approach [6]. The cross-validation accuracy is equal to the root mean square error (RMSE) of all the v runs. Figure 3 shows the root mean square error surface for ϵ=0.5 with different (γ, C) combinations during f_{CQ} regression. Various combinations of (ϵ, C, γ) values are tried and the one with the best cross-validation accuracy (i.e. the lowest RMSE) is selected. .

IV. Experimental results

A. Experimental setup and training data preparation

We use Booksim2.0 [7] to simulate the router and NoC performance on 4×4 and 8×8 meshes. Various synthetic traffic patterns including *uniform random*, *transpose* and *tornado*

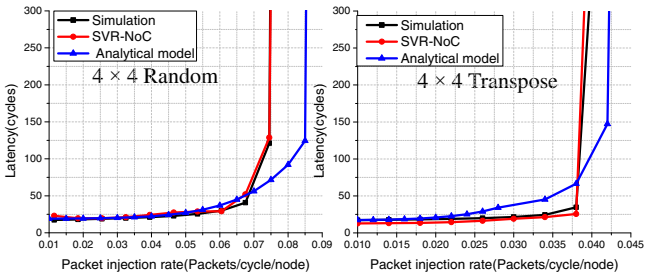

Figure 4. Comparison with analytical model

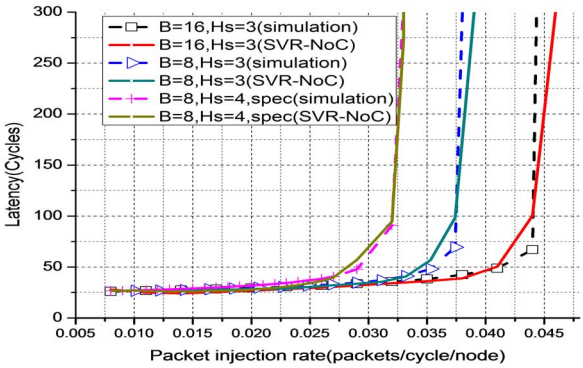

Figure 5. Latency prediction for trained traffic patterns

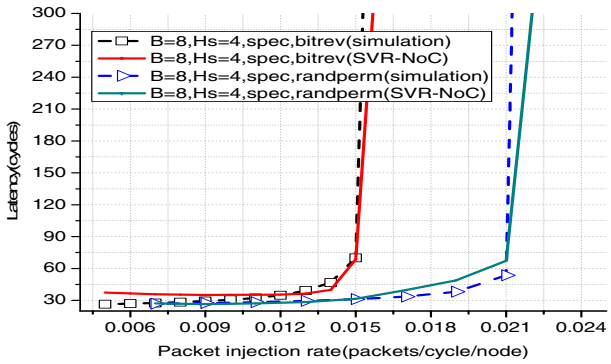

Figure 6. Latency prediction for untrained synthetic traffics

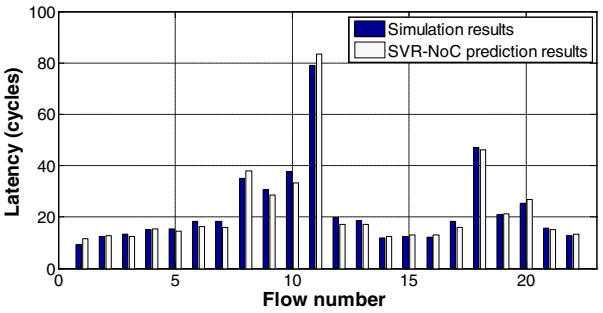

Figure 7. Flow latency comparison for MMS benchmark

[8] are used as inputs to the target router architecture to obtain the training data-set. The training data set contains fifty different runs under each traffic pattern, which provides sufficient training data for the SVR learning engine as well as reasonable training time.

B. Experimental results

1) SVR-NoC for synthetic traffics and real applications : We first compare the SVR-NoC with state-of-the-art analytical model proposed in [2]. In this comparison, 4×4 meshes with random and transpose traffic are considered. We assume a constant data packet length of six flits and the depth of all channel buffers is equal to eight flits in the simulation. As shown in Figure 4, the analytical model incurs more than 10% error in predicting the network criticality. On the other hand, the SVR-NoC can predict the network saturation point very accurately, with less than 3% error for both traffic patterns.

Next, we show the SVR-NoC performance for the trained and un-trained traffic patterns in Figure 5 and 6. Figure 5 shows the simulations results for random traffics under various pipeline depth H_s and buffer depth B combinations as well as the non-speculative and speculative router architectures [8]. Figure 6 shows the results for two new synthetic traffic patterns *(bitreversal*, *random permutation*). The SVR-NoC demonstrates high accuracy in both cases with less than 5% error in predicting the network criticality status. We also use a real benchmark, multimedia application MMS [9] for the comparison and demonstrate the accuracy of SVR-NoC in Figure 7. As shown in Figure 7, the maximum difference between the simulation results and the SVR-NoC regression results is less than 10% and the average error is about 4.5%.

2) Runtime comparison: The average simulation time on an 8×8 mesh for 1×10^6 cycles is about fifteen minutes while the SVR-NoC predictive framework takes about eight seconds. The latency estimation method using an analytical model [2] takes about six seconds. We can see that the SVR-NoC has a similar run-time with the analytical model and has a $120X$ speedup over the simulation-based prediction.

V. Conclusion

In this work, we propose a machine-learning based approach for NoC performance prediction. We demonstrate the modeling accuracy by using independent test data sets. The SVR-NoC model achieves less than 10% error for various traffic patterns with about 120X speedup compared to the simulation-based estimation.

References

[1] A. Hemani, A. Jantsch, S. Kumar, A. Postula, and J. Oberg, "Network-on-chip: An architecture for billion transistor era," in *proc. IEEE NorChip Conference*, nov. 2000.

[2] M. Lai, L. Gao, N. Xiao, and Z. Wang, "An accurate and efficient performance analysis approach based on queuing model for network on chip," in *proc. ICCAD 2009.*, nov. 2009, pp. 563 –570.

[3] A. E. Kiasari, Z. Lu, and A. Jantsch, "An analytical latency model for networks-on-chip," *Very Large Scale Integration (VLSI) Systems, IEEE Trans. on*, vol. PP, no. 99, pp. 1 –11, 2012.

[4] U. Ogras, P. Bogdan, and R. Marculescu, "An analytical approach for network-on-chip performance analysis," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Trans. on*, vol. 29, no. 12, pp. 2001 –2013, dec. 2010.

[5] V. Vapnik, *Statistical Learning theory*. John Wiley and Sons, 1998.

[6] C. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

[7] *Booksim 2.0, http://nocs.stanford.edu/booksim.html*, 2012.

[8] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2003.

[9] J. Hu and R. Marculescu, "Energy- and performance-aware mapping for regular noc architectures," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Trans. on*, vol. 24, no. 4, pp. 551 – 562, april 2005.