Rede Paralela Dedicada a Serviços para Tolerância a Falhas em MPSoCs baseados em NoC

Augusto Erichsen†, Eduardo W. Wächter *†, Fernando G. Moraes † (Advisor)
†FACIN – Pontifical Catholic University of Rio Grande do Sul – PUCRS – Av. Ipiranga 6681– Porto Alegre – RS – Brazil
*Department of Informatics – University of Santa Cruz do Sul – Santa Cruz do Sul – RS – Brazil
{augusto.erichsen,eduardo.wachter}@acad.pucrs.br, fernando.moraes@pucrs.br

Abstract – *Mechanisms for runtime fault-tolerance in MPSoCs are mandatory to cope with transient and permanent faults. This issue is even more relevant in nanotechnologies due to process variability, aging effects, and susceptibility to upsets, among other factors. The literature presents isolated solutions to deal with faults in the MPSoC communication infrastructure. In this context, one gap to be fulfilled is to integrate all layers, resulting in a solution to cope with NoC faults from the physical layer up to the application layer. The goal of this work is to present a runtime integrated approach to cope with NoC faults in MPSoCs. The original contribution is the proposal of a set of hardware and software mechanisms to ensure both efficient and reliable communication in NoC-based MPSoCs. The proposal has an acceptable silicon area overhead and a small memory footprint. Experiments with benchmarks reveal a worst-case execution time overhead of 6.5%, and the average-case lower than 0.5%. Thousands of scenarios with random fault injection were simulated and all of them were executed correctly.*

Keywords – NoC-based MPSoC, fault-tolerant NoCs, fault-tolerant communication, fault recovery.

1. INTRODUÇÃO

O *gap de produtividade* se refere ao fato de que o número de transistores num chip cresce mais que a habilidade de projeta-los e integra-los. Uma possível solução para atacar o gap de produtividade é o reuso de componentes, solução esta que tem sido utilizada nos últimos anos e resultou em Sistemas em Chip (System-on-Chip, SoC) e recentemente em Sistemas Multiprocessados em Chip (Multi Processor SoC).

MPSoCs são SoCs que reusam elementos de processamento (*Processing Elements*, PEs), interconectados através de um barramento ou Redes Intra-Chip (*Network on Chip*, NoC). Devido a razões de escalabilidade, NoCs são utilizadas em arquiteturas MPSoC. A possibilidade de integrar em um futuro próximo, milhares de núcleos em um chip [1] apresenta desvantagens: a variabilidade do processo e os efeitos do envelhecimento tornam-se proeminentes em nanotecnologias, levando ao aumento de falhas transientes e permanentes quando comparadas às tecnologias antigas.

De outro lado, a adoção de MPSoCs baseados em NoCs prove uma excelente oportunidade de projeto para sistemas confiáveis. Por exemplo, existem caminhos redundantes numa NoC, assim como módulos redundantes (PEs). Estas características justificam a utilização destas arquiteturas com propósito de tolerância a falhas (*Fault Tolerance*, FT): continuar a execução corretamente apesar da existência de falhas ao custo de uma penalidade no desempenho.

Este trabalho utiliza uma infraestrutura pronta de um MPSoC baseado em NoC para fins de FT. Este trabalho tem dois objetivos principais: (i) Utilizar uma rede de serviços para detecção de pacotes perdidos numa NoC e (ii) Comparar duas abordagens para armazenamento dos caminhos sem falha entre dois processadores.

2. Plataforma MPSoC de Referência

Um MPSoC consiste de um conjunto de PEs interconectados por uma rede intrachip. O PE utiliza uma API de comunicação com duas primitivas do tipo MPI: uma função *Enviar()* não-bloqueante e uma *Receber()* bloqueante. Para implementar uma troca de mensagens, um espaço de memória do OS reservado, denominado *pipe*, é utilizado para armazenar as mensagens trocadas pelas tarefas.

A Figura 1 detalha o protocolo de comunicação usado para mensagens trocadas entre tarefas. A execução de um *Enviar()* transfere o conteúdo da mensagem para uma posição livre do pipe, essa posição é marcada como usada (*USED*). Quando a tarefa B (mapeada no PE_2) executa um *Receber()*, um pacote de requisição de mensagem (*MSG REQUEST*) é enviado ao PE_1. A tarefa B é então bloqueada (*WAITING*). Quando o PE_1 recebe a requisição de mensagem, ele executa duas ações: (i) a mensagem no pipe passa para o estado de espera de confirmação de recebimento (*WAITING_ACK*); (ii) a mensagem é enviada ao PE_2. Ao receber a mensagem, o PE_2 transfere seu conteúdo para a memória da tarefa B e habilita a tarefa para execução e envia uma mensagem confirmando o recebimento da mensagem. Somente após receber esta confirmação, a posição do pipe é liberada.

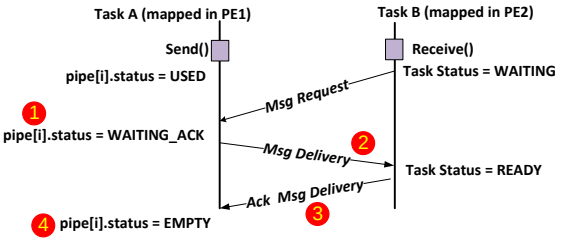


Figura 1 – Protocolo de comunicação entre pacotes de dados.

Este trabalho utiliza recursos comumente encontrados em NoCs [7]: topologia malha 2D, encaminhamento de pacotes por tunelamento, algoritmo de roteamento adaptivo e canais físicos duplicados. Características relacionadas à FT incluem:

(i) Suporte ao roteamento distribuído e roteamento na origem;
(ii) Células de isolamento nas portas de entrada dos roteadores;
(iii) Rede Paralela de Serviços, que pode ser utilizada para serviços de monitoramento e descoberta de caminhos entre roteadores.

2.1 Envio de mensagens no MPSoC Referência

A troca de mensagens entre processadores no MPSoC HeMPS é feita através de uma interface hardware – software.

O processador executa um OS com um protocolo de comunicação baseado em MPI. Por outro lado, os módulos de hardware NI e DMA são responsáveis pelo auxílio do envio de mensagens para a rede. Na prática, a última camada do protocolo de comunicação é responsável por programar e disparar o módulo de DMA. O módulo de DMA lê a memória diretamente e escreve no módulo NI, liberando o processador para continuar a execução do código da tarefa. Desta forma, pode-se executar computação (no processador) e comunicação (módulos DMA e NI) de forma paralela.

A Figura 2 exemplifica o envio de um pacote de rede contendo uma mensagem do PE1 para o PE2. A tarefa no PE1 executa uma chamada de OS, o qual executa uma rotina para montagem do pacote de rede a ser transmitido na memória (Figura 2 – 1). Após, o OS programa o módulo de DMA para que ele execute o envio deste pacote (Figura 2 – 2) e dispara o envio. O DMA por sua vez acessa a memória diretamente (Figura 2 – 3) e repassa o pacote à NI (Figura 2 – 4). A NI é responsável por traduzir as mensagem de 32 bits vindas do processador no tamanho do flit da NoC (neste caso, 16 bits) (Figura 2 – 5).

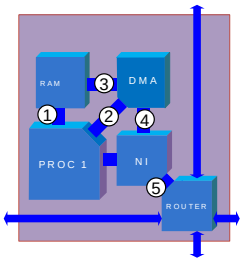


Figura 2 – Arquitetura do Elemento de Processamento. Os números identificam o processo de envido de uma mensagem do PE ao roteador.

3. TRABALHOS RELACIONADOS

Esta seção avalia os trabalhos recentes relacionados a: (*i*) FT em nível de links [3]; (*ii*) FT em nível de roteador [4]; (*iii*) FT em nível de algoritmo de roteamento [5]; (*iv*) FT em nível de API de comunicação [6]. A pesquisa de Radetzki [2] contem um compreensivo estado-da-arte em métodos de FT para NoCs. Veiga e Zeferino [3] usa duas técnicas de FT em nível de links. CRC e paridade. Nos dois casos, o erro é detectado em nível de flit, precisando de retransmissão de flit. O trabalho visa falhas transientes, devido a erros de crosstalk. O Autor informa um overhead de áre de 6.25% (paridade) a 28% (CRC). Esta abordagem não é acoplada a uma solução de recuperação de falhas para o sistema. Chang et al. [4] propôs uma técnica com roteadores redundantes para aumentar a confiabilidade. A técnica corrige uma falha permanente para cada coluna da NoC, que pode ocorrer nos links ou roteadores. A técnica consiste em adicionar um roteador no topo de cada coluna de uma malha 2D. Quando uma falha é detectada, a NoC é reconfigurada, sem modificar a topologia original. Este método suporta um número limitado de falhas uma vez que há um número limitado de roteadores sobressalentes.

Rodrigo et al. [5] apresenta o uLBDR, uma arquitetura de roteador para roteamento FT em NoCs. Ele guarda o estado de conectividade dos roteadores vizinhos, provendo caminhos mínimos. O uLBDR propôs a replicação dos pacotes em duas portas de saída, garantindo alcance completo, porém aumentando a congestão da rede. Aulwes e Daniel [6] incluiu recursos de confiabilidade na MPI como ckecksum, retransmissão de mensagens, e re-roteamento de mensagens.

Os trabalhos analisados melhoraram a confiabilidade do MPSoC limitadamente. A maioria das abordagens suporta um número limitado de falhas [3-5] ou serve para aplicações limitadas [6]. A *contribuição* do presente trabalho reside em um método de FT em camadas para MPSoCs. A camada de teste não é acoplada a um método para uma NoC específica, então o projetista pode escolher a melhor abordagem de teste para aplicação. Na camada de reconfiguração, o mecanismo de descobrimento de caminhos é rápido e trabalha com múltiplas falhas na NoC. Na camada de aplicações, o código fonte não é modificado. É importante mencionar que este trabalho foca na camada de reconfiguração de falhas da NoC. Métodos de teste de NoCs e PEs tolerantes a falhas não são abordados.

4. EXEMPLO MOTIVACIONAL

Considere na Figura 3 uma falha no caminho entre as tarefas A e B. Usando uma biblioteca convencional de troca de mensagens, sem mecanismos de FT, uma chamada *Receber()* pode esperar indefinidamente se uma camada de comunicação insegura perder a mensagem. Ainda, se um pacote é atingido por uma falha ele pode ser mal roteado ou ocupar *buffers* indefinidamente e um roteador falho pode gerar sinais incorretos para seus vizinhos (falha bizantina). Portanto, para evitar falhas no sistema causadas por falhas de hardware, o sistema deve localizar, evitar e isolar a região falha. Por exemplo, se a falha é permanente e a rede utiliza apenas algoritmo de roteamento determinístico, então a retransmissão do pacote usaria o mesmo caminho falho, gerando um novo pacote pendente. Assim, o sistema deve ter a capacidade de *localizar o roteador falho* e ter suporte a *roteamento adaptativo*, enviando pacotes usando um caminho alternativo *evitando* a região falha.

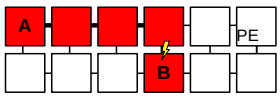


Figura 3 – Exemplo motivacional para justificar a adoção de um protocolo de comunicação FT. Em vermelho o caminho assumido em um algoritmo de roteamento XY.

5. REDE PARALELA DE SERVIÇOS

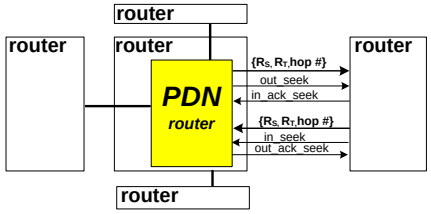
Esta seção apresenta a Seek Network, uma rede paralela dedicada a executar serviços em NoCs. Estes serviços podem ser de:

- Monitoramento: A rede entrega a PEs específicos sobre a ocupação de canais;
- Congestão: A rede entrega a PEs específicos informações sobre caminhos bloqueados devido a congestionamentona rede.
- Tolerância a Falhas: A rede entrega ao PE origem que a mensagem passou por um nodo falho e por isso não será entregue ao destino.

Como esta rede executa paralelamente à rede de dados, ela não interfere na comunicação. A rede paralela propaga seus dados como uma onda, um *broadcast* distribuído (por isso o nome da rede *seek*), onde um PE fonte passa a mensagem para os roteadores vizinhos, e estes por sua vez repassam a informação para seus vizinhos. Cada PE visitado pela mensagem guarda os dados em uma pequena tabela temporária, para verificar se já não transmitiu aquela informação. Isto evita que a mensagem se propague infinitamente. Esta rede envia mensagens de apenas 1 *flit*,

evitando a utilização de buffers.

A Figura 4 ilustra a arquitetura da rede paralela. Ela é composta por roteadores que se comunicam entre si. Para utilização dos serviços, temos os sinais in/out_seek para informar o envio do serviço e o tipo deste, in/out_ack_seek para confirmar o recebimento do serviço e a tupla {R_S, R_T, hop#}, que são o PE fonte e o PE destino respectivamente e Hop# que é o contador do número de saltos até a mensagem chegar no destino. Um salto é a medida utilizada para distância entre um roteador e outro.



R_S – adress of the source PE
R_T – adress of the target PE
out_seek = {null, seek, clean, backpropagate, fault}

Figura 4 – Roteador da Rede Paralela de Serviços, com sua interface externa.

Quando o PE fonte envia uma mensagem pela rede *Seek* ele configura o sinal out_seek com o serviço desejado, preenche os campos origem e destino e zera o contador de saltos. Cada roteador possui uma pequena tabela para armazenar os serviços que estão trafegando na NoC e informações como os PEs origem e destino do serviço.

O primeiro serviço da rede paralela se chama *Seek*. A função deste serviço é propagar uma onda de um PE a outro, como ilustrado na **Erro! Fonte de referência não encontrada.**. Quando o *Seek* é recebido, a porta que o recebeu e os valores da tupla são armazenados na tabela. Então, cada roteador da rede Seek repassa a tupla {R_S, R_T, hop#+1} para seus vizinhos, exceto para a porta que recebeu a requisição. Assim a onda é propagada para todos os PEs. Se um PE receber a requisição e o R_S já estiver armazenado na tabela, a requisição é descartada. Isso garante que o processo de propagação pare sem nenhum controle complexo, após todos os PEs serem visitados. O segundo serviço da rede Seek é o *Clean*. Este serviço é similar ao *Seek*, no sentido de sua propagação, porém seu objetivo é limpar a tabela após ter sido utilizada.

O terceiro serviço é chamado *BackPropagate*. O R_T injeta na NoC (não na rede Seek) um pacote que segue a rota inversa percorrida pelo serviço de busca. Funciona direcionando o pacote para a porta que foi anotada anteriormente na tabela. O *BackPropagate* termina quando o R_S é alcançado.

O quarto serviço é chamado *Fault* e é transmitido para o R_S quando um pacote de dados ou de controle alcança uma porta falha. Este serviço é propagado da mesma forma do *Seek* e sua função é informar para o roteador origem que ele deve buscar um novo caminho livre de falhas para o R_T.

5.1 TOLERÂNCIA A FALHAS

O presente trabalho tem como um dos objetivos a utilização da rede Seek para FT. O objetivo é utilizar a rede Seek para notificar uma mensagem não entregue devido a uma falha no caminho até o destino através de um protocolo de comunicação tolerante a falhas. Para utilizar uma abordagem deste tipo, necessitamos de uma pilha de camadas para assegurar comunicações confiáveis. Esta pilha inclui:

- método de teste para testar as portas ou o roteador inteiro;
- wrapper cells para isolar regiões falhas e sinalizar falhas;
- roteamento adaptativo para lidar com caminhos falhos;
- suporte a nível de SO para computar novos caminhos, para retransmitir pacotes e para descartar pacotes incompletos;

A Figura 5 apresenta um exemplo de comunicação com falhas. O PE2 solicita ao PE1 uma mensagem (1 na Figura 5). O PE1 envia uma mensagem com destino ao PE2 (2 na Figura 5), entretanto, o PE vizinho a este está falho (PE com falha na porta oeste) e a mensagem não será entregue. O PE falho está isolado (através da utilização de wrappers que estão fora do escopo do trabalho). O PE vizinho (PEff) ao encaminhar uma mensagem ao PE falho, dispara o processo de notificar o PE1 que a mensagem não foi entregue.

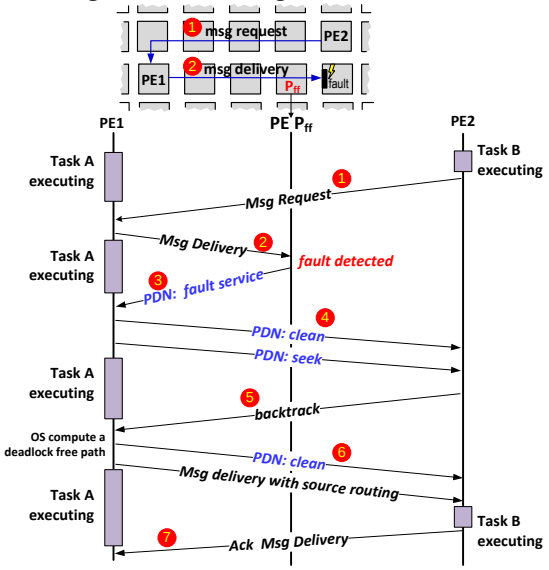


Figura 5 – Método de recuperação de falhas.

O PEff descarta todos os flits recebidos que seriam encaminhados para a porta falha e utiliza a rede Seek para enviar o serviço *Fault* (3 na Figura 5). Quando o PE1 recebe o serviço *Fault* o processador é interrompido e duas ações são tomadas pelo SO: (*i*) o serviço *Clean* é enviado para limpar as tabelas usadas na onda *Fault* anterior; (*ii*) é enviado um *Seek* para encontrar o PE2 (4 na Figura 5).

Ao ser alcançado pelo *Seek*, o PE2 utiliza a NoC para enviar o serviço *BackPropagate* (5 na Figura 5) que irá conduzir o caminho livre de falhas descoberto pelo *Seek*. O caminho retornado para este exemplo pode ser N-L-L-L-L (1 salto para norte e 4 saltos para leste). No passo 6 da Figura 5 o pacote de *BackPropagate* chega ao PE1 e este envia o serviço *Clean* para limpar os dados utilizados pelo *Seek* para encontrar o caminho. O caminho é armazenado no PE1 e este então reenvia o pacote que não pode ser entregue devido à falha. No passo 7 o PE2 envia a confirmação de recebimento da mensagem e escalona a tarefa B para execução.

6. Tabelas com Caminhos Livres de Falha

O PE é o módulo responsável pelo armazenamento dos caminhos livres de falhas entre os pares comunicantes na rede. O segundo objetivo deste trabalho é a comparação de duas abordagens para armazenamento destes caminhos: uma com a tabela armazenada na NI e outra com a tabela armazenada em software pelo OS. As duas abordagens se diferem basicamente

no local onde os caminhos livres de falhas (recebidos da rede seek) são armazenados.

A abordagem com tabela na NI armazena estes caminhos em uma memória dedicada com 8 posições. A vantagem deste método é que o overhead para envio das mensagens é mínimo, pois a consulta na tabela e envio para a NoC é feita pela NI. Como desvantagens temos o número limitado de posições (destinos dos pacotes) e o overhead de área causado pela tabela. A abordagem por software é feita de maneira diferente. A NI contém apenas uma tabela com uma posição. Ao receber os caminhos pelo serviço *BackPropagate*, estes são enviados ao OS e armazenados em uma estrutura. Toda vez que o OS for enviar uma mensagem, ele verifica nesta estrutura se há um caminho armazenado para o destino. Se houver, ele escreve na NI o caminho a ser percorrido. A lógica para envios da NI não é alterada entre as versões com tabela em Software ou Hardware.

7. RESULTADOS

A Tabela 1 apresenta os resultados de desempenho do protocolo de comunicação tolerante a falhas (Figura 5), variando a distância entre o processador origem (PE_1) até o processador destino (PE_2). Esta tabela compara a abordagem com tabelas implementadas em software (SW) e no módulo Network Interface (NI). A aplicação utilizada para medida é um decodificador MPEG, com mensagens de 128 palavras (pacotes de 256 flits) do PE_1 para o PE_2. Não existe congestionamento na rede, visto que o objetivo é a caracterização do protocolo.

Tabela 1 – Número de ciclos de clock para cada passo do protocolo de comunicação FT com tabela na NI ou SW

hop #	t2 → t3		t3 → t4		t4→ t5		t5 → t6		t6 → t7		TOTAL	
	NI	SW	NI	SW	NI	SW	NI	SW	NI	SW	NI	SW
2 hops	769	777	647	647	48	48	981	981	1023	1031	**3468**	**3484**
3 hops	769	777	672	672	51	51	1084	1084	1023	1031	**3599**	**3615**
4 hops	769	777	688	688	63	63	1177	1177	1023	1031	**3720**	**3736**
5 hops	769	777	704	704	75	75	1291	1291	1023	1031	**3862**	**3878**
6 hops	769	777	720	720	87	87	1402	1402	1023	1031	**4001**	**4017**
7 hops	769	777	736	736	99	99	1499	1499	1023	1031	**4126**	**4142**

Na Tabela 1:

- t2→t3 – Tempo para injetar um pacote na NoC e receber a detecção da falha. Este tempo inclui todos os passos discutidos na seção 2.1. Basicamente o tempo é independente do número de hops pois o número de ciclos para transmitir e receber a mensagem pela rede é menos que o tempo gasto na NoC.
- t3→t4 – Tempo entre a detecção da falha pelo PE_1 e a recepção de um SEEK no PE_2. O OS no PE_1 é interrompido pelo pacote de detecção de falha, disparando o serviço de clean e logo após o serviço de SEEK para o PE_2. Tipicamente há um aumento de 16 ciclos de clock por hop. Neste caso não há diferença entre os resultados pois as tabelas não influenciam nestes valores.
- t4→t5 – Tempo para transmitir o pacote de backtrack. Devido a implementação em hardware, um pequeno número de ciclos de clock é necessário e este valor é uma função do número de ciclos de clock.
- t5→t6 – Corresponde ao número de ciclos para computer um novo caminho no OS. Também é uma função do número de hops.
- t6→t7 – Tempo gasto para entregar os pacotes utilizando o caminho livre de falhas. No caso da tabela em software inclui o tempo para verificar se deve ou não escrever o caminho na tabela da NI. O número de ciclos de clock é constante devido à mesma razão explicada na parcela t2→t3.

Nos cenários sem falhas, o tempo necessário para transmitir um pacote é tipicamente 1700 ciclos de clock (Figura 6) para a rede com tabelas na NI e 1770 ciclos para a rede com tabela em software. Estes valores se justificam devido à necessidade de verificar a cada envio se o OS deve escrever um caminho na NI.

Entretanto, o overhead de pior caso é observado no cenário de 7 hops, correspondendo a 2,6 vezes o tempo para transmitir a mensagem sem falhas. A transmissão da primeira mensagem é rápida devido à armazenagem de mensagens enquanto a tarefa consumidora inicia a execução. A terceira mensagem requer reconfiguração do caminho e retransmissão de mensagem. Todas outras mensagens não são penalizadas, pois o novo caminho também é mínimo e o tempo para transmitir a mensagem em ambos os casos é o mesmo. Consequentemente, o impacto de todos os passos de reconfiguração da Figura 5 é mínimo.

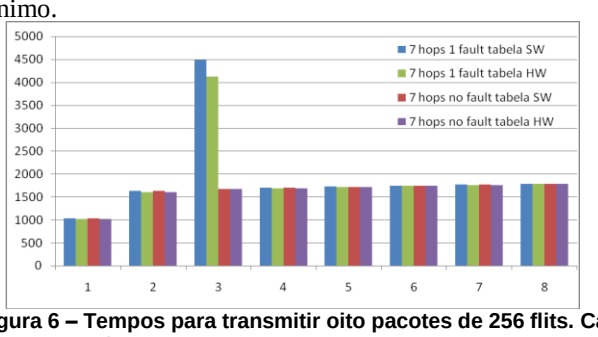


Figura 6 – Tempos para transmitir oito pacotes de 256 flits. Cada transmissão é dividida em quatro configurações: (i) sem falhas na tabela em software, (ii) sem falhas com tabela na NI, (iii) com falha com tabela em software e (iv) com falha com tabela na NI.

8. CONCLUSÃO

Até onde o Autor pode verificar, este é o primeiro trabalho integrando todas as camadas necessárias para lidar com falhas na NoC, garantindo uma correta execução das aplicações sem modificar o código fonte original. As falhas são tratadas em tempo de execução, com uma penalidade de desempenho desprezível para a maioria dos casos.

Um recurso relevante do trabalho proposto é a transparência para o desenvolvedor de aplicações. O método reside em um esquema de reconfiguração de caminhos (implementado em hardware) acoplado ao suporte do SO. O método ainda possibilita reduzir o custo de área de silício da NI, causando um overhead mínimo na comunicação.

9. REFERENCIAS

[1] Borkar, S. Thousand Core Chips - A Technology Perspective. In: DAC, 2007, pp.746-749.

[2] Radetzki, M.; Feng, C.; Zhao, X.; Jantsch, A. *Methods for Fault Tolerance in Networks on Chip*. ACM Computing Surveys, v.46(1), 2014, 36p. (pre-print online available)

[3] Veiga, F.; Zeferino, C. *Implementation of Techniques for Fault Tolerance in a Network-on-Chip*. In: WSCAD-SCC, 2010, pp. 80-87.

[4] Chang, Y.; Chiu, C; Liu, S.; Liu, C. *On the Design and Analysis of Fault Tolerant NoC Architecture Using Spare Routers*. In: ASP-DAC, 2011, pp. 431-436.

[5] Rodrigo, S.; *et al. Cost-Efficient On-Chip Routing Implementations for CMP and MPSoC Systems*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, v. 30(4), pp. 534–547, 2011.

[6] R. Aulwes and D. Daniel. *Architecture of LA-MPI, a Network-Fault-Tolerant MPI*. In: IPDPS, 2004, 6p.

[7] Agarwal, A.; Iskander, C.; Shankar, R. *Survey of Network on Chip (NoC) Architectures & Contributions*, Journal of Engineering, Computing and Architecture, v.2(1), 2009.