UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE TECNOLOGIA - DEPARTAMENTO DE INFORMÁTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO II

ANDERSON DARIO ARENDT

PERFORMANCE DE REDES MESH EM CHIP SOB DIFERENTES CONFIGURAÇÕES: UMA ANÁLISE UTILIZANDO SIMULAÇÃO FULL-SYSTEM

MARINGÁ

2019

ANDERSON DARIO ARENDT

PERFORMANCE DE REDES MESH EM CHIP SOB DIFERENTES CONFIGURAÇÕES: UMA ANÁLISE UTILIZANDO SIMULAÇÃO FULL-SYSTEM

Trabalho apresentado como requisito parcial para a obtenção do título de bacharel no curso de Ciência da Computação pela Universidade Estadual de Maringá.

Orientador: Prof. Dr. Anderson Faustino Da Silva

MARINGÁ

2019

TERMO DE APROVAÇÃO

ANDERSON DARIO ARENDT

PERFORMANCE DE REDES MESH EM CHIP SOB DIFERENTES CONFIGURAÇÕES: UMA ANÁLISE UTILIZANDO SIMULAÇÃO FULL-SYSTEM

Trabalho apresentado como requisito parcial para a obtenção do título de bacharel no curso de Ciência da Computação pela Universidade Estadual de Maringá, pela seguinte banca examinadora:

Prof. Dr. Anderson Faustino Da Silva
Orientador

Prof. Dr. Ronaldo Augusto de Lara Gonçalves
UEM

Prof. Me. Nilton Luz Queiroz Junior
UEM

Maringá, Agosto de 2019.

RESUMO

Nos últimos anos várias técnicas para tornar os processadores mais eficientes foram desenvolvidas. Com a popularização dos processadores *multicores* e *manycores* surgiu o conceito de redes-em-chip (NoCs). Esse conceito tenta prover uma maior escalabilidade e eficiência aos processadores trazendo consigo conceitos de redes de computadores para serem implementados em chip. Entretanto, diversas questões devem ser levadas em conta na hora de se trabalhar com NoCs. Neste trabalho apresentamos um estudo sobre NoCs – em especial as com topologia *Mesh*, avaliando o seu impacto sob diferentes configurações em sistemas completos, utilizando simulação *full-system*. Com esse tipo de simulação é possível obter dados mais fidedignos do real comportamento de uma NoC. Utilizando um pacote de aplicações, foram realizados simulações e então avaliado o impacto da variação de parâmetros como topologia, protocolos de coerência de cache, algoritmos de roteamento, dimensão da rede, dentre outros, do ponto de vista de vazão, latência, média de saltos das mensagens, consumo de energia e ocupação média das filas nos roteadores. Além do mais também foi realizado, em uma das aplicações utilizadas durante a simulação, a implementação de três métodos de balanceamento de carga, visando observar a influência exercida por esses diferentes métodos no desempenho de uma NoC.

Palavras-chaves: mesh. noc. full-system.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A natureza e complexidade das aplicações modernas demandam cada vez mais desempenho computacional. Por muito tempo uma estratégia para prover um maior desempenho por parte dos processadores se deu aumentando as suas frequências de *clock*, possibilitando assim executar mais instruções em menos tempo. Entretanto, não foi possível continuar aplicando essa estratégia por muito tempo. Questões como dificuldade de dissipação de energia acabaram se tornando um sério problema, tanto do ponto de vista energético quanto na capacidade dos componentes manterem sua integridade operando sob altas temperaturas.

Com tais limitações, projetistas viram a necessidade de recorrer a diferentes técnicas. Começou então a ideia de adicionar níveis de paralelismo aos processadores, inicialmente a nível de instruções, com a criação do conceito de *pipelines* e replicação destes, criando assim os chamados processadores superescalares. Porém, segundo Borkar (2003), aumentar a complexidade do circuito elétrico não trás um ganho de desempenho considerável comparado com o investimento feito. Para contornar esse problema de custo-benefício surgiram os processadores *multicore*. Processadores *multicore* consistem em uma técnica de agrupar vários processadores no mesmo chip – onde cada processador é chamado de núcleo.

O nível cada vez maior de paralelismo nos processadores impacta diretamente na evolução, não somente de paradigmas de programação, mas também na necessidade constante de evolução de sua arquitetura e organização interna (MA et al., 2014). Em processadores com mais de 8 núcleos, os chamados *manycores* torna-se muito difícil integrar esses núcleos baseando suas conexões por meio de barramentos. Portanto, para que essa estratégia de colocar mais núcleos em um chip continue sendo viável, a arquitetura interna dos processadores deve ser capaz de prover uma boa escalabilidade, e assim surgiu o conceito de redes-em-chip (NoC's).

Redes-em-chip utilizam a ideia de redes de computadores para conectar os componentes de um processador. Utilizando componentes chamados de roteadores, é possível criar uma rede onde os eles serão os responsáveis por deliberar mensagens para os núcleos conectados a essa rede. As NoC's melhoram diversos aspectos de desempenho de um processador, como latência e taxa de transferência entre núcleos, entretanto ela deve ser bem projetada para que isso ocorra. Segundo Ma et al. (2014), essa camada adicional de comunicação entre núcleos composta pelos roteadores chega a ser responsável em alguns casos pelo consumo de cerca de 36% da energia consumida pelos processadores. Logo, escolher uma boa topologia, bons protocolos de comunicação e coerência de cache tornam-se cruciais para que o processador

tenha um bom desempenho computacional e também energético (CHEN; GILLARD; C. LI, 2012) (TEDESCO et al., 2005) (GARDEA et al., 2017) (SAINI; AHMED, 2015) (HAO et al., 2011) (Q. YANG; WU, 2010) (CHANG; CHIU, 2012) (HOLSMARK; KUMAR, 2005) (PSATHAKIS et al., 2015).

Neste trabalho foi realizado um estudo sobre redes *Mesh* em chip a fim de investigar o impacto que diferentes configurações causam em um sistema completo. Essa pesquisa foi motivada pelo fato de não haver muitos trabalhos sobre o impacto de redes-em-chip em sistemas completos. A topologia *Mesh* foi escolhida por existir uma grande quantidade de estudos abordando-a de várias formas diferentes, e também pelo fato dela ser majoritariamente na literatura uma escolha comparativa com as demais topologias existentes ou propostas.

2 REVISÃO BIBLIOGRÁFICA

Nesta seção é apresentado uma breve revisão bibliográfica sobre conceitos chaves para o entendimento do trabalho desenvolvido.

2.1 TOPOLOGIA MESH EM REDES EM CHIP

Topologia refere-se como os nós de uma rede estão dispostos, ou seja, seu posicionamento no espaço. Sanchez, Michelogiannakis e Kozyrakis (2010) comentam que a escolha de uma topologia impacta diretamente no desempenho de uma rede. Isto ocorre porquê cada topologia possui um conjunto de caminhos possíveis diferentes para as mensagens que trafegam na rede, e portanto, ocorre diferentes padrões de tráfego.

Segundo Nychis et al. (2010), Mesh é a topologia mais popular, a qual é implementada em diversos *manycores* comerciais e protótipos de pesquisa. Esse fato se dá pela sua simplicidade e as vantagens que essa topologia trás. A Figura 1 mostra o esquema de uma rede *Mesh*, onde a letra R denota roteadores, e C as unidades de processamento. Cada nó roteador possui no máximo 5 interfaces de entrada e saída, onde 4 são para comunicação entre os roteadores vizinhos e uma é para a interface de rede, a qual conecta o roteador com a unidade de processamento. Entretanto hoje em dia também há o conceito de *Mesh Concentrada (CMesh)*, que são redes *Mesh* onde há mais unidades de processamento conectadas ao mesmo roteador, visando manter uma baixa média de números de saltos na rede (H. J. KIM; SEO; HAN, 2011).

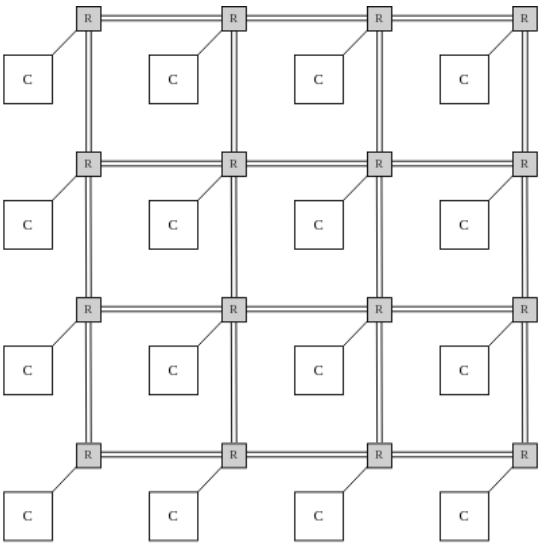


FIGURA 1 – Topologia Mesh

Nychis et al. (2010) apontam como vantagens das redes *Mesh* a fácil escalabilidade, baixa latência de comunicação nó-a-nó e a fácil implementação de algoritmos de roteamento. Por outro lado, existe uma desvantagem, também apontada por Nychis et al. (2010), nos casos em que uma mensagem tem que percorrer um grande número de roteadores para chegar ao seu destino final, isso porquê cada salto de uma mensagem de um roteador ao outro inclui uma latência, e cada passagem por um roteador agrega uma latência de processamento nodal, isso acaba impactando diretamente o consumo de energia devido a existência de mais estágios de bufferização, transmissão e controle da mensagem. Portanto, a escolha de parâmetros como, algoritmo de roteamento, protocolo de coerência de cache, disposição e quantidade de caches e controladores de memória, tamanho do *link* dentre outros deve ser cuidadosa para que se minimize essa desvantagem.

2.1.1 Algoritmos de Roteamento

O algoritmo de roteamento é uma das escolhas mais importantes no projeto de uma NoC pois ele é responsável por determinar o caminho que a mensagem terá de percorrer até chegar ao seu destino. Algoritmos que possuem essa função podem ser divididos segundo Palesi e Daneshtalab (2014) com base nas decisões de roteamento em dois grupos: *source-routing* e *distributed-routing*.

Source-routing: nesse tipo de algoritmo, o roteador raiz (o primeiro a receber a mensagem) já define todo o caminho que a mensagem terá que percorrer. Algoritmos com essa característica possuem algumas desvantagens, por exemplo: (i) pode haver concorrência entre *links*, isto é, caso um *link* esteja sendo utilizado para transmitir uma mensagem A e este *link* também é necessário para a transmissão da mensagem B, a mensagem B terá que esperar esse *link* ficar disponível, não podendo escolher outro *link* para seguir seu caminho; (ii) se há k roteadores entre a origem e o destino da mensagem, a mensagem deve possuir em seu *header* a informação da rota a ser seguida, ou seja, informação sobre os k roteadores que compõe o caminho, acarretando em mensagens com *payloads* cada vez maiores.

Distributed-routing: algoritmos distribuídos fazem com que as mensagens tenham em seu *header* apenas o endereço de origem e destino, fazendo que a rota seja decidida dinamicamente em cada roteador que a mensagem passa. Aqui pode haver mais três classificações baseando-se em como a rota e determinada. São chamado de *determinísticos* quando a rota é definida levando em conta a posição relativa em relação a origem e destino, sendo indicados para redes com padrão de tráfego regular. Por outro lado, há os chamados de *adaptativos*, onde a rota é determinada levando em conta a condição da rede, sendo assim indicados para redes que possuem padrão de tráfego irregular. E por fim os

parcialmente-adaptativos que em determinadas situações adotam um comportamento adaptativo, já em outras um comportamento determinístico.

2.1.2 Protocolos de Coerência de Caches

Processadores modernos utilizam memórias caches para minimizar a necessidade de leituras e escritas de dados direto na memória principal do computador. A existência de vários níveis de cache em processadores multicores e *manycores* torna necessária a existência de mecanismos para manter os dados coerentes. Isso ocorre porquê diferentes núcleos em algum momento podem conter cópias do mesmo dado da memória principal em suas respectivas caches. Sendo assim, quando esse dado é modificado em um núcleo os outros devem saber que essa modificação foi realizada a fim de não haver inconsistência nos cálculos realizados com este dado (HENNESSY, 2013). O mecanismo responsável por manter os dados compartilhados entre diferentes caches coerentes é chamado de protocolo de coerência de cache.

Conforme explicam Al-Waisi e Agyeman (2017), a ideia principal dos protocolos é informar os outros núcleos que um dado foi modificado em uma cache local. Sendo assim, quando os outros núcleos forem informados que o dado que eles possuem na sua respectiva cache foi modificado por outro núcleo, eles devem buscar atualizar esse dado de alguma forma.

Há dois grandes grupos de protocolos, os baseado em invalidação e os baseados em atualização. Os protocolos baseados em invalidação utilizam a ideia de exclusividade sobre um dado, tornando a escrita serializável. Quando uma escrita vai ser realizada em um dado compartilhado, as cópias desse dado existente em outros núcleos são imediatamente invalidadas, sendo assim, somente quando esses núcleos precisam ler esse dado ocorre um *cache-miss*, forçando-os buscar uma cópia atualizada do dado. Já os protocolos baseados em atualização fazem que, quando um dado é modificado em um núcleo, uma a cópia dele atualizada seja enviada via *broadcast* para todos os núcleos. Essa transmissão em *broadcast* do dado atualizado para todos os núcleos consome muita largura de banda, fazendo assim que os protocolos baseado em invalidação sejam os mais utilizados (PATTERSON; HENNESSY, 2008).

2.1.3 Controladores de Memória

Aumento de núcleos por chip inevitavelmente aumenta a quantidade de acessos a memória, e caso a forma de comunicação entre processador e memória não seja eficiente isso pode se tornar um gargalo no desempenho do processador como um todo. Xu, Liljeberg e Tenhunen (2011) explicam que, utilizar um canal de comunicação duplo ou até mesmo triplo, embora duplique ou triplique a largura de banda de comunicação

entre memória e processador, acaba sendo uma estratégia custosa, pois requer mais módulos de memória e pode gerar maior consumo de energia e aumentar.

Uma alternativa para otimizar a comunicação processador-memória é utilizar vários controladores de memória distribuídos pela NoC, onde cada controlador fica responsável por controlar o acesso a uma faixa de endereços físicos da memória principal. Entretanto, seu posicionamento na rede deve ser pensado a fim de minimizar a distância da mensagem entre memória e núcleo requisitante do dado, e também visando distribuir uniformemente as mensagens entre os nós da rede sem causar pontos de congestionamento e, consequentemente, degradação de desempenho (XU; LILJEBERG; TENHUNEN, 2011).

2.1.4 Links

As conexões entre componentes da rede são chamadas de *links*. Elas são canais de comunicação *full-duplex* compostos por um conjunto de fios. As mensagens que devem ser trasportadas pela rede são divididas em *flits*, que são pequenos pacotes com o tamanho igual a largura do *link*. Quanto maiores e mais velozes forem os *links* melhor será o desempenho da rede (COTA; MORAIS AMORY; LUBASZEWSKI, 2011).

2.1.5 Canais Virtuais

Visando aumentar o fluxo da rede e diminuir o risco de *deadlocks* algumas redes podem implementar o conceito canais virtuais. Essa técnica é aplicada nos roteadores da rede, transformando um único canal de comunicação em vários canais virtuais através da multiplexação lógica, onde cada canal virtual contém seus próprios *buffers* (COTA; MORAIS AMORY; LUBASZEWSKI, 2011).

2.2 SIMULAÇÃO FULL-SYSTEM

Simulação é uma etapa importante em diversos processos. Ela consiste em, a partir de um modelo de um sistema, realizar experimentos sobre ele. Assim como é possível construir modelos climáticos para observar a formação de padrões e a evolução no clima ocorrem, ou também modelos de colisão para atestar a segurança de um veículo, é possível criar modelos de computadores completos, isto é, um modelo que contenha em sua representação processadores, periféricos, memórias, e redes de conexão e toda a interface de comunicação entre aplicação, sistema operacional e hardware, tornando possível realizar simulações a fim de observar as mais diversas questões comportamentais. Simulação utilizando modelos de computadores completos são chamados de *Full-System Simulation* (ENGBLOM, 2019).

O uso de *Full-System Simulation* é realizado em diversos cenários. Além de ser usado para observar o desempenho e comportamento de uma arquitetura particular, ele é utilizado também para: (i) desenvolvimento e teste de sistemas complexos ou de alto risco; (ii) simular integração de um novo modelo de hardware com o computador; (iii) atuar como ambiente de desenvolvimento na produção de softwares de baixo-nível; (iv) realizar testes e injeção de falhas em sistemas e aplicações dentre outros usos (ENGBLOM, 2019).

Existem diversos simuladores de sistemas completos, um comparativo entre os mais famosos pode ser visto na Tabela 1.

Simulator	**Processor Archtectures Suported**	**Accuracy**
Sismics	Alpha, ARM, MIPS, PowerPC, SPARC and x86	Functionally-accurate
PTLsim	x86	Cycle-accurate
SimpleScalar	Alpha, ARM, PISA and x86	Cycle-accurate
OVPsim	Open Cores, Open RISC, ARM, Synopsys, ARC, MIPS, PowerPC, Xilinx, MicroBraze others	Instruction-accurate
GEM5	Alpha, ARM, MIPS, PowerPC, SPARC, RISCV and x86	Cycle-accurate

TABELA 1 – Comparativo entre simuladores

Para este trabalho foi escolhido o simulador GEM5 para realização dos experimentos. Esse simulador foi construído de forma modular e flexível, permitindo explorar e avaliar diversos aspectos arquiteturais. Por tais características, por ser de código aberto e por ter estudos prévios, como o realizado por Butko et al. (2012) no qual atestam sua estabilidade e acurácia, faz com que ele seja amplamente utilizado pela comunidade científica.

O GEM5 é um simulador construído nas linguagens Python e C++. Atualmente ele oferece dois modos de simulação, que são:

System Emulation (SE): neste modo não há a camada de sistema operacional, havendo assim somente interação por meio de *system-calls*.

Full System (FS): este modo simula um sistema completo. Portanto, dados não somente da arquitetura mas também referente ao sistema operacional passam a ser monitorados.

Tal simulador também oferece várias escolhas de CPUs, como o *AtomicSimple, TimingSimple, InOrder* e *DerivO3*. Parâmetros referente ao processador podem ser facilmente modificados e algoritmos previsores de desvio podem ser implementados e integrados. Há também a opção de escolher entre diversas arquiteturas de processador. Quanto ao sistema de memória há dois, que são o *Classic Model*, herdado do simulador

M5, e o *Ruby Model*, que oferece suporte a diferentes protocolos de coerência de cache e também possui maior acurácia nos resultados (BUTKO et al., 2012). Além disso, o *GEM5* também oferece o *Garnet Network Model*, que é um modelo responsável por representar interconexões de rede dentro do simulador, tornando possível realizar simulações de NoC's (AGARWAL et al., 2009).

3 CONFIGURAÇÃO DOS EXPERIMENTOS

Os experimentos foram realizados no *GEM5* em modo *Full-System* e consistiram em variar os seguintes parâmetros em uma rede *Mesh* 4x4: protocolo de coerência de cache, algoritmos de roteamento e quantidade, posicionamento de controladores de memória, tamanho dos *flits*, entre outras.

3.1 APLICAÇÕES

Para avaliar o comportamento da rede sob diferentes configurações foi utilizado o pacote de aplicações *CAP Bench*, pois esse pacote possui aplicações destinadas a avaliação de processadores multicores e *manycores*, ideal para o objetivo deste estudo (SOUZA et al., 2017). As aplicações escolhidas para realização dos experimentos são as seguintes:

- **FAST:** Algoritmo de processamento que identifica regiões de interesse em uma imagem.
- **FN:** Algoritmo que procura pares de números amigos em um dado intervalo de entrada. Um número é dito amigo de outro se seu valor é igual a soma de todos os divisores de outro e vice versa.
- **GF:** O GF consiste em aplicar filtro de suavização em imagens por meio de operações matriciais.
- **IS:** Aplicação baseada no *bucket-sort* para ordenação de um conjunto de números inteiros.
- **KM:** É um algoritmo para o popular problema das *k-means* que consiste em dividir um conjunto de pontos em partições e encontrar k pontos nesse conjunto de forma que esses pontos sejam o centro de suas respectivas partições.
- **LU:** Algoritmo de fatoração de matrizes utilizado em resolução de sistemas e para encontrar matrizes inversas.
- **TSP:** Aplicação para o problema do caixeiro-viajante que consiste em fazer um viajante passar por n cidades apenas uma vez e retornar a sua origem pelo menor caminho possível.

Visando investigar o impacto do balanceamento de carga, foram desenvolvidos três versões para o algoritmo FN com balanceamentos distintos, onde os detalhes de cada implementação podem ser vistos no Apêndice.

Na Tabela 2 é mostrado quais os tamanhos de entrada do *dataset small* para cada aplicação bem como o método de paralelização que cada um utiliza.

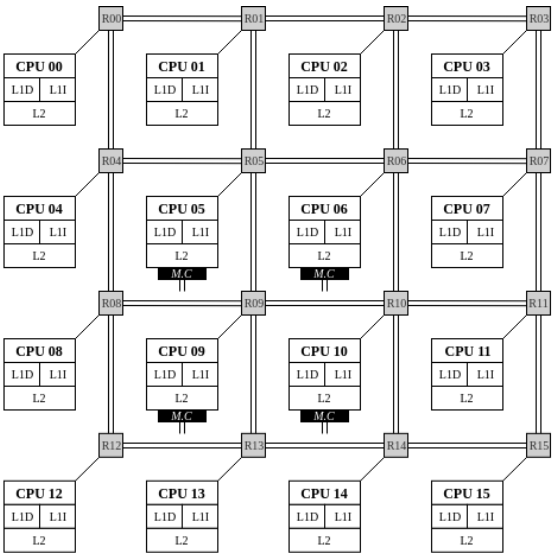
App	Strategy	Dataset
FAST	Stencil	4096×4096
FN	MapReduce	$8 \times 10^6 + 1$ to $8 \times 10^6 + 2^{13}$
GF	MapReduce	4096×4096 (image) 7×7 (mask)
IS	Divisão e conquista	2^{24} integers
KM	Map	$2^{13} R^{16}$ points, 512 centroids
LU	Workpool	1024 x 1024
TSP	Workpool	15 towns

TABELA 2 – Detalhes das aplicações escolhidas, Souza et al. (2017)

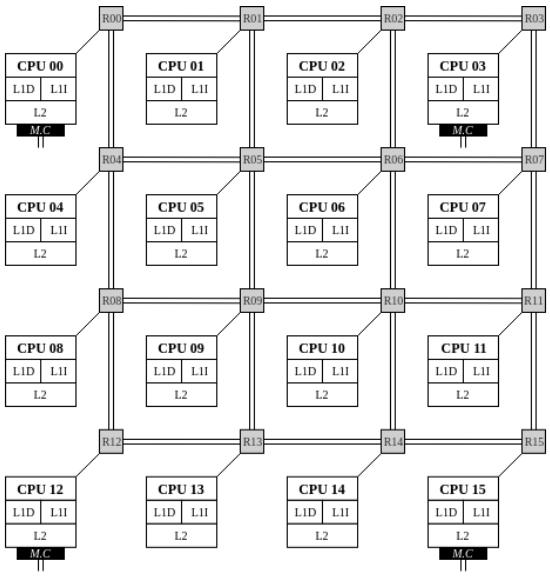
3.2 CONTROLADORES DE MEMÓRIA

Quanto as variações de configurações da rede, primeiramente, iremos destacar a questão dos controladores de memória presentes nela. A Figura 2 ilustra as variações do uso de controladores de memória.

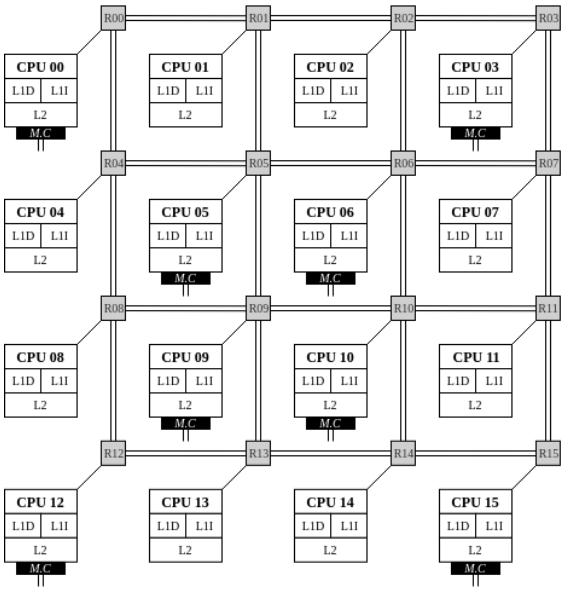
Como a rede é 4x4 existem 16 CPUs na rede, cada uma possuindo cache L1 separada para dados e instruções, e uma cache L2 unificada. Os dispositivos denotados pela letra R são os roteadores, enquanto os denotados pelas iniciais MC são os controladores de memória.



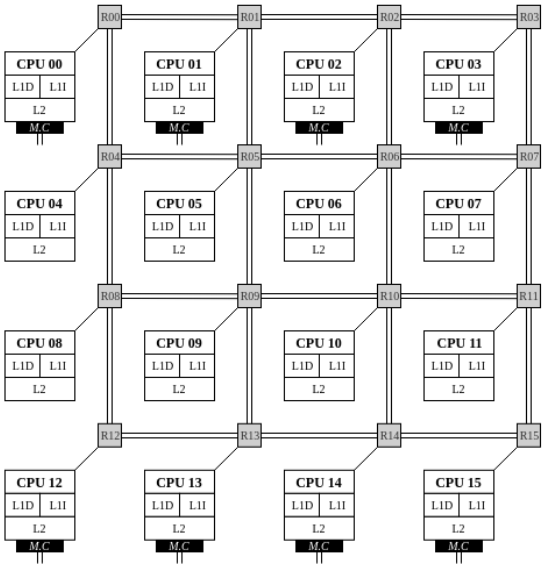
(a) Middle



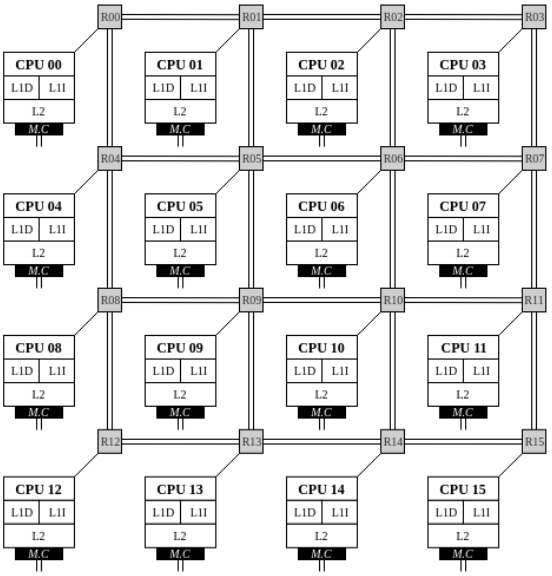
(b) Corners



(c) Middle-corners



(d) 2-rows



(e) All

FIGURA 2 – Disposição de controladores

Há duas configurações onde ambas possuem 4 controladores de memória porém dispostas de forma diferente, que são a *middle* e a *corners*. Foram elaboradas também duas configurações com 8 controladores de memória e outra com 16 controladores.

3.3 PROTOCOLOS DE COERÊNCIA DE CACHE

Foram adotados três tipos de protocolos de coerência de *cache*, que são fornecidos pelo *GEM5*, que são o *MOESI Hammer, MOESI CMP Directory* e *MESI CMP Directory*. Tanto o protocolo MESI quanto MOESI em suas versões CMP Directory

usam o princípio de invalidação de estados (AL-WAISI; AGYEMAN, 2017).

Os respectivos algoritmos podem ser vistos como autômatos onde, cada letra em seu nome é um estado com um respectivo significado, havendo regras específicas para transição entre os estados. No caso desses dois protocolos suas iniciais denotam os seguintes significados:

- **M:** usado para indicar que uma linha foi modificada, ou seja, seu valor difere do valor contido na memória principal, sinalizando ao processador que esse dado terá que ser atualizado na memória principal em algum momento, porém antes de que outra unidade realize uma leitura.
- **E:** estado usado para indicar que a o dado está presente somente na cache atual e seu valor coincide com o da memória principal.
- **S:** indica que um dado existe em mais de uma cache e portanto a respectiva cache deve passar a escutar sinais de invalidação vindo de outras.
- **I:** usado para indicar que o dado está inválido na cache atual.
- **O:** estado presente no protocolo MOESI, é usado para suprir a demanda pelo dado a outras caches quando essas possuem cópias do dado e seu respectivo valor na memória principal está desatualizado.

Tais algoritmos podem ter seguir duas estruturas de implementações. A implementação por *CMP Directory* é uma versão que possui uma estrutura de diretório para manter armazenada informações de dados privados nas *caches*. Por outro lado a versão *Hammer*, que foi proposta pela AMD, não possui uma estrutura de controle, fazendo necessário um uso intensivo de comunicação em *broadcasts* para os processadores (ROS; E.; M., 2010).

Nos experimentos utilizamos 4 versões do *MOESI Hammer*, que são: (0) padrão; (1) com compartilhamento migratório de blocos (STENSTRÖM; BRORSSON; SANDBERG, 1993); (2) com *probe-filter* (CONWAY et al., 2010); e (3) com *full-bit directory* (BRYANT; BECKMANN, 2012).

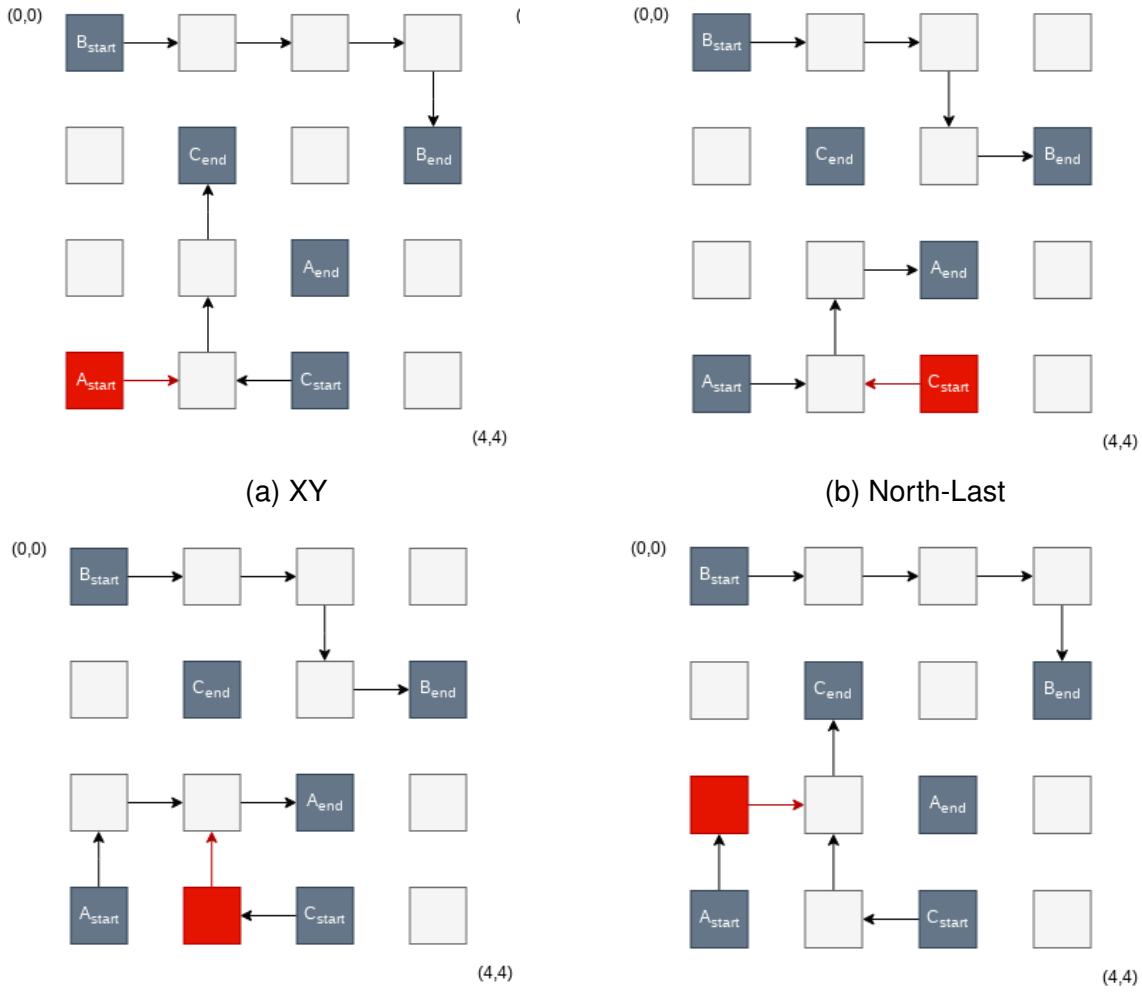
3.4 ALGORITMOS DE ROTEAMENTO

Conforme comentado anteriormente, a escolha do algoritmo de roteamento é um dos parâmetros mais importantes para um bom desempenho da rede. Em nossos experimentos são avaliados 4 algoritmos de roteamento, os quais são descritos abaixo.

XY: É um algoritmo determinístico cujo funcionamento consiste em rotear as mensagens primeiramente pelo eixo x e, quando a coordenada corrente x coincidir com

a coordenada x destino o pacote passa a ser roteado pelo eixo y. A Figura 3a ilustra o funcionamento deste algoritmo. Por se tratar de um algoritmo determinístico, pacotes podem ter que esperar um roteador em seu caminho que esteja em uso ficar livre para prosseguir, é o que acontece com a mensagem B, onde a cor vermelha denota o caminho bloqueado (PALESI; DANESHTALAB, 2014).

North-last: Pertencente a classe dos algoritmos parcialmente adaptativos, esse algoritmo funciona de forma determinística se a condição de coordenadas $y_{end} \leq y_{start}$ acontecer (Figura 3b caminho A e C), caso contrário ele irá rotear as mensagens de forma adaptativa (Figura 3b caminho B). A fim de evitar *deadlocks*, essa classe de algoritmo impõe algumas restrições em sentidos de roteamento das mensagens. No caso do *north-last* é proibido as mensagens realizarem duas conversões ao mesmo sentido quando estão sendo trafegadas para a direção norte.



(a) XY

(b) North-Last

(c) Negative-First

(d) West-First

FIGURA 3 – Algoritmos de Roteamento

Negative-first: Também é um algoritmo parcialmente adaptativo. Primeiramente ele

tenta seguir em alguma direção negativa. Caso ocorra de $(x_{end} \leq x_{start} \wedge y_{end} \geq y_{start}) \vee (x_{end} \geq x_{start} \wedge y_{end} \leq y_{start})$ então a mensagem é roteada de forma determinística (Figura 3c caminho B), caso contrário, de forma adaptativa (Figura 3c caminho A e C). Neste algoritmo a restrição para evitar *deadlocks* é evitar duas conversões consecutivas em um sentido negativo.

West-first: Algoritmo parcialmente adaptativo onde sempre que ocorrer a condição $(x_{end} \leq x_{start})$ para uma mensagem, então essa será transmitida de forma determinística (Figura 3d mensagem A e C), caso contrário de forma adaptativa (Figura 3d caminho B). Nesse, é proibido que duas mensagens realizem duas conversões consecutivas para a direção oeste.

3.5 OUTROS PARÂMETROS

Por fim, além da variação dos parâmetros já descritos anteriormente, na Tabela 3 é mostrado as demais configurações adotadas na realização dos experimentos.

Parametro	**Valor**
CPU	TimingSimple
Clock	2GHz
L1I Size	32kb
L1I Assoc	4
L1D Size	32kb
L1D Assoc	4
L2 Size	128kb
L2 Assoc	8
Memory Size	1GB
Link Size	64, 128, 256 (kb)
Virtual Channels	2, 4, 8

TABELA 3 – Parametros da NoC.

Foi escolhido uma configuração *baseline* que serve como base para as variações de parâmetro de execução. Todas as variações que serão apresentadas consistem de a alteração do respectivo parâmetro na configuração *baseline*. Tal configuração é mostrada na Tabela 4 abaixo.

Parâmetro	Valor
Tamanho da rede	4x4
Topologia C.M.	Middle-corners
Algoritmo de roteamento	XY
Protocolo C. Cache	MESI CMP_Directory
Tamanho de link	128 kb
Canais Virtuais	4
Dataset	small

TABELA 4 – Parâmetros *baseline*

Além de avaliarmos como diferentes configurações em uma rede *Mesh* impactam no desempenho de aplicações, também avaliamos como diferentes topologias se comportam com tais aplicações. Para isso foram escolhidas, com parâmetros semelhantes a configuração *baseline*, três topologias a fim de comparar com a topologia *Mesh*, que são: *Crossbar*, *FatTree* e *Torus* onde, tais topologias utilizam o princípio de tabela de roteamento (BO WANG et al., 2014).

4 RESULTADOS

Nessa seção são apresentados os resultados dos experimentos, onde foram avaliados os seguintes pontos: (1) vazão da rede; (2) taxa média de dados trafegados pelos controladores; (3) número médio de saltos dos pacotes; (4) latência média dos pacotes nos canais virtuais; (5) taxa de ocupação das filas dos roteadores; e (6) energia média consumida por segundo. Por uma falha de execução não foi possível obter dados referente a execução do algoritmo LU para uma escalabilidade 8x4 e de IS para o protocolo MOESI-Hammer3.

4.1 VAZÃO DA REDE

A primeira forma de avaliação do impacto das variações de configurações é quanto a vazão da rede. Tal vazão é data em *bits/s*, calculada utilizando dados de total de pacotes trafegados na rede e tempo de execução da aplicação. Os resultados são mostrados na Figura 4, onde a escala é definida como sendo k denotando mil, m milhão e b bilhão.

Com a variação de *dataset* notou-se que a vazão varia muito de aplicação para aplicação. Houve casos como o GF em que o aumento do *dataset* e o aumento da vazão da rede são diretamente proporcionais. Enquanto também há casos como no caso do TSP em que essa relação é inversamente proporcional. Mas na maior parte dos cenários indicam que existe um fator limitante em que, a partir de um *dataset small* a vazão não muda. Quanto ao caráter topológico, há uma situação que o desempenho foi visivelmente melhor que outras, que são os casos da *baseline* e *Torus* com o GF, e também o TSP.

Alterar a disposição dos controladores de memória na rede só teve um grande impacto no algoritmo IS, onde as disposições *corners* e *2-rows* tiveram um desempenho muito inferior comparado com a disposição *middle* e *middle-corners*. Já quanto a questão topológica, a aplicação que foi mais influenciada pelas mudanças foi a GF, onde as topologias *Crossbar* e *FatTreee* tiveram um desempenho pior que as demais. Nos demais algoritmos não houve impacto considerável diante das variações propostas.

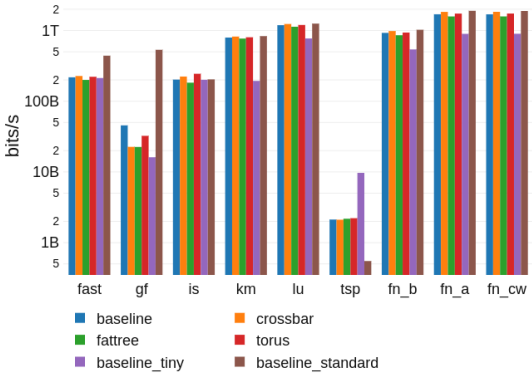
Em todas as aplicações com exceção de IS, constatou-se que a variação da largura do *link* de comunicação implica diretamente no na vazão da rede, quanto maior o *link* maior a vazão.

Quanto aos protocolos de coerência de cache, os destaques positivos são os protocolos MOESI-Hammer0 e MOESI-Hammer1 que foram dominantes em 7 aplicações. O protocolo MESI teve um desempenho dominante, nas suas versões CMP

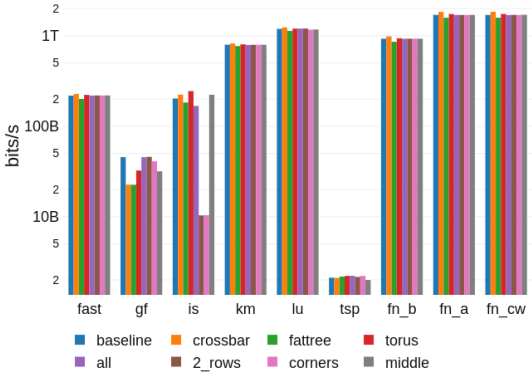
Directory e HAMMER em KM. Observou-se que em aplicações em que os protocolos da família MESI dominam os da família MOESI obtiveram um desempenho inferior e vice versa.

Quanto ao roteamento só influenciaram no desempenho das aplicações GF e IS, nos demais o desempenho não sofreu mudança considerável.

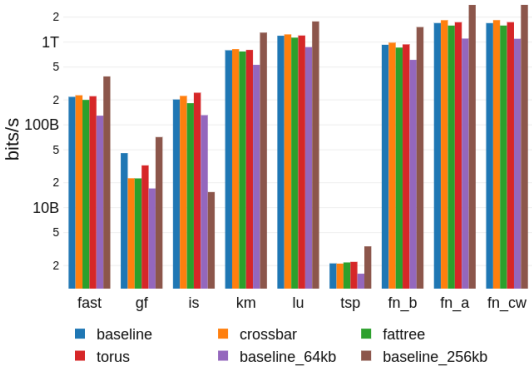
Aumentar o tamanho da rede apenas impactou consideravelmente positivamente quando na proporção 4x8 as aplicações GF e TSP. Por outro lado numa proporção 2x4 o desempenho foi pior que a rede 4x4 em 7 aplicações.



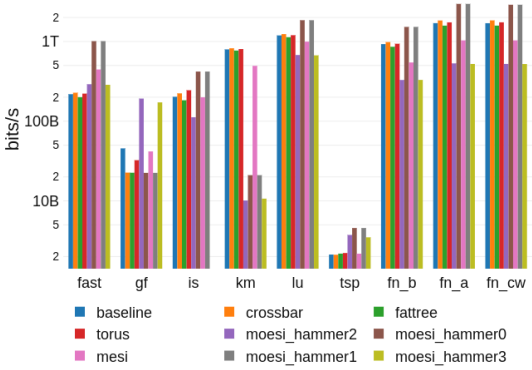
(a) Dataset



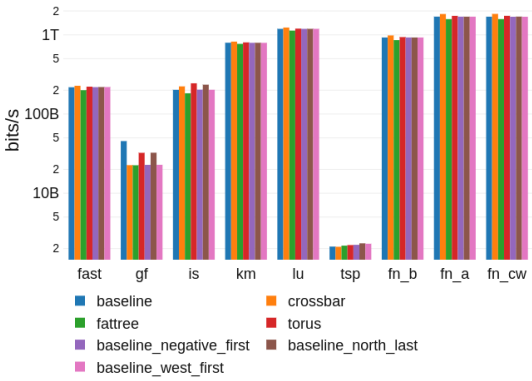
(b) Controlador de memória



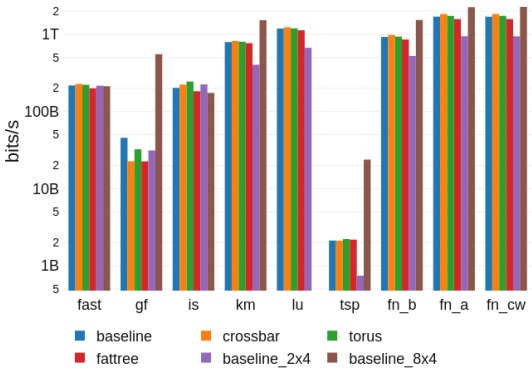
(c) Link



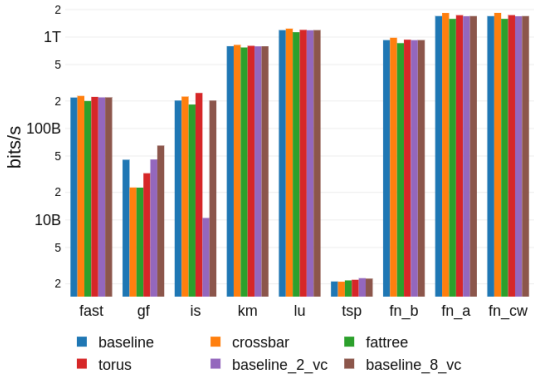
(d) Protocolo



(e) Algoritmo de roteamento



(f) Escalabilidade



(g) Canais Virtuais

FIGURA 4 – Vazão da rede

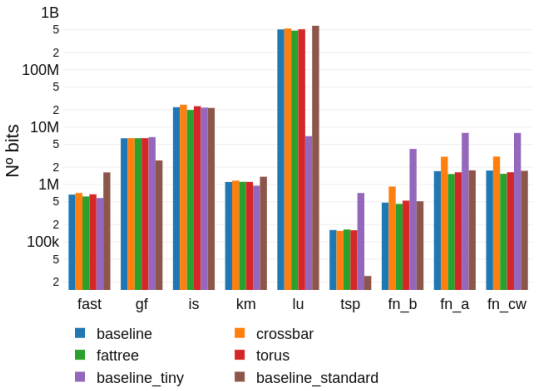
O número de canais virtuais apenas teve um impacto interessante em GF, onde o comportamento com dois ou quatro *(baseline)* obtiveram desempenho semelhante, entretanto quando com 8 o desempenho tem uma leve melhora. Já em IS diminuir a quantidade de canais virtuais provoca uma queda considerável no desempenho.

4.2 TRÁFEGO MÉDIO NOS CONTROLADORES

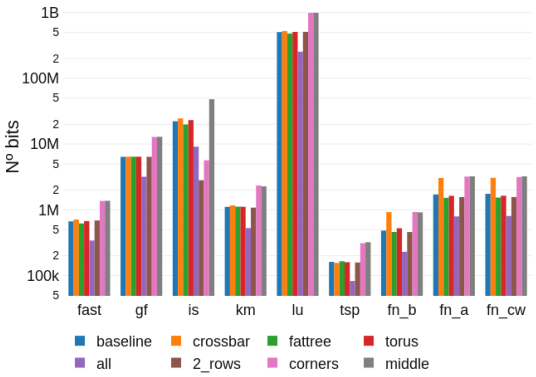
Foi avaliado também o quanto as diferentes versões da rede impactam no desempenho de seus controladores de memória. Neste caso foi optado por avaliar o tráfego médio em bits nos controladores. Os resultados são mostrado na Figura 5.

Neste quesito de avaliação com exceção do TSP a topologia *Crossbar* dominou os cenários.

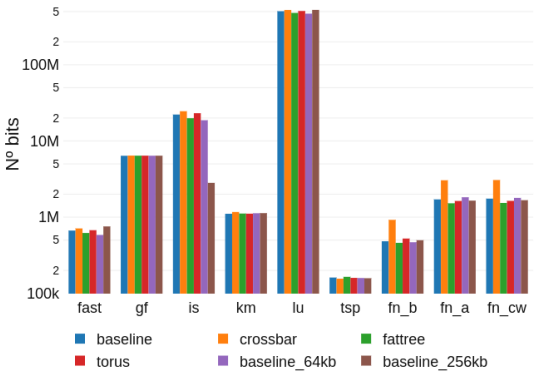
Quanto ao *dataset* mais uma vez há diversos comportamentos. Houve casos como o do TSP e dos FN's em que um *dataset* menor fazia com que mais fosse utilizado os controladores. E também casos em que quanto maior, maior o uso dos controladores (FAST e KM) e também casos que a não houve variação considerável.



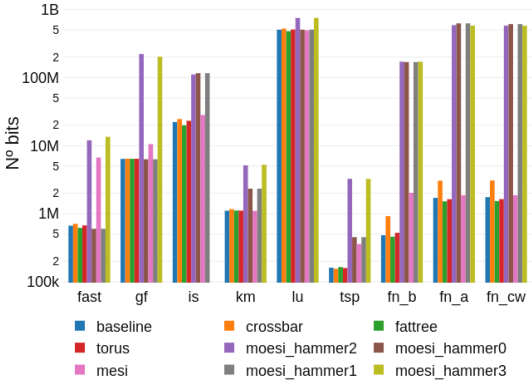
(a) Dataset



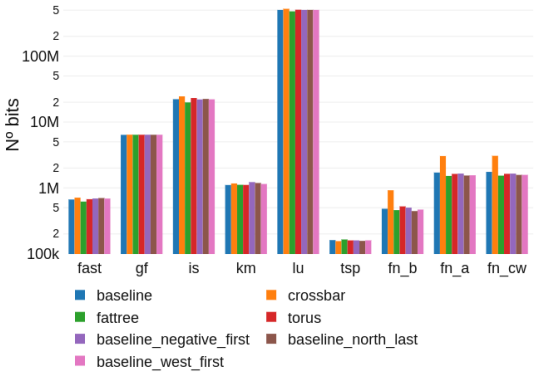
(b) Controlador de memória



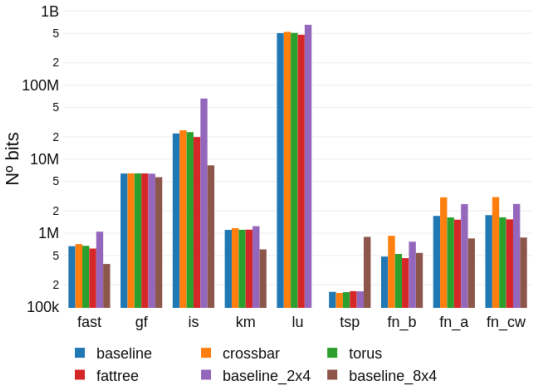
(c) Link



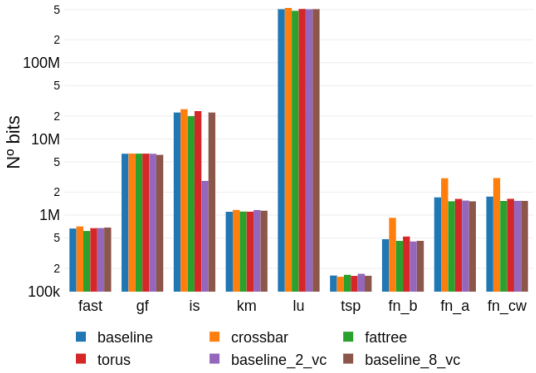
(d) Protocolo



(e) Algoritmo de roteamento



(f) Escalabilidade



(g) Canais Virtuais

FIGURA 5 – Tráfego médio dos controladores

No cenário de disposição de controladores de memória, foi observado que a configuração *all* teve um uma média menor de uso em todos os cenários com exceção de IS. Quanto a topologia a que exigiu mais dos controladores foi a *FatTree*.

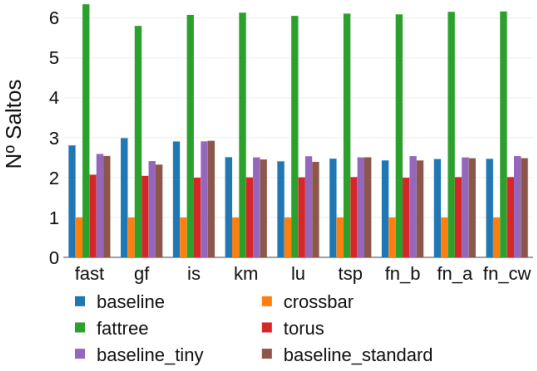
A variação nos tamanho de link bem como o algoritmo de roteamento não impactaram de forma considerável o uso dos controladores.

Avaliando a escalabilidade, observou-se que a disposição 2x4 a há um uso maior que as demais, porém, quando com 4x8 o uso dos controladores diminuiu. Entretanto com TSP esse padrão não acontece.

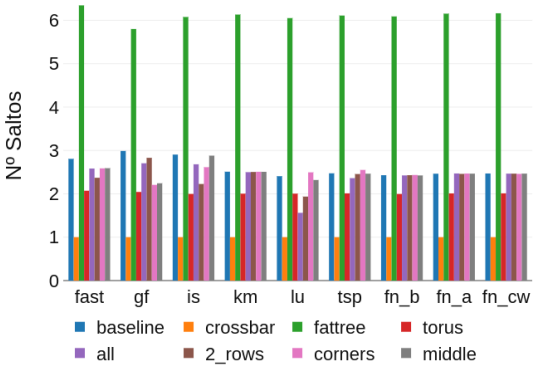
Já na questão de canais virtuais, a alteração deles só impactou quando variada negativamente em relação ao *baseline* no caso do IS.

4.3 SALTOS

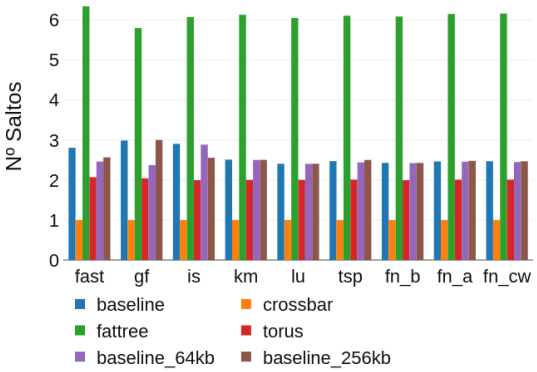
A quantidade de saltos em uma rede também é um fator que determinante para avaliar se ela está sendo eficiente. A Figura 6 mostra os dados obtidos referente a média de saltos necessários para um pacote chegar ao seu destino.



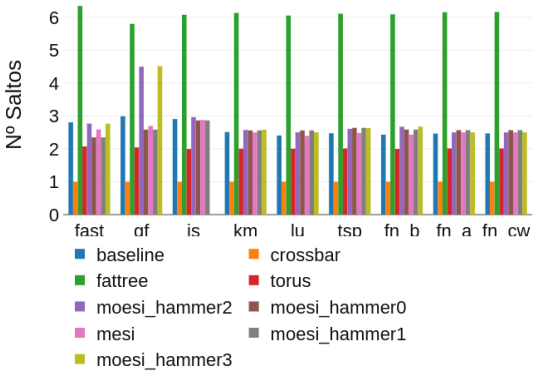
(a) Dataset



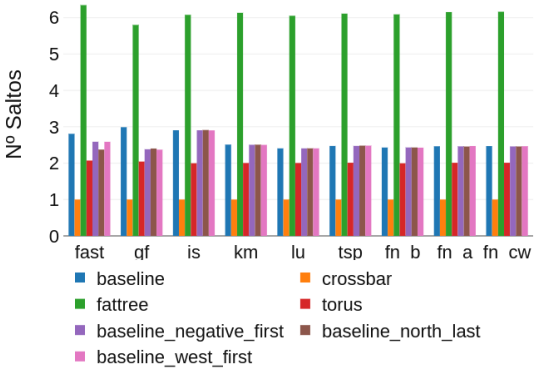
(b) Controlador de memória



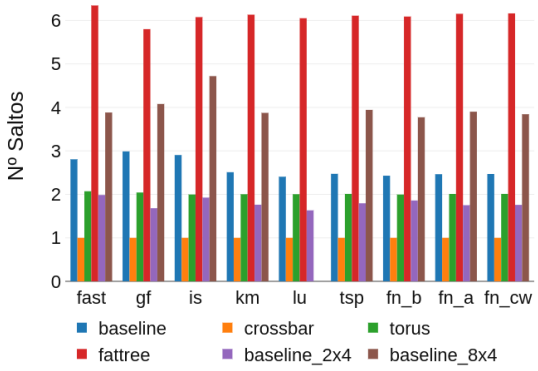
(c) Link



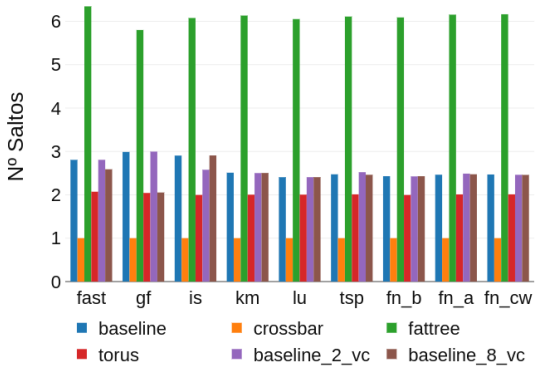
(d) Protocolo



(e) Algoritmo de roteamento



(f) Escalabilidade



(g) Canais Virtuais

FIGURA 6 – Média de saltos de pacotes

A topologia com melhor desempenho nesse quesito foi a obviamente a *Crossbar* pelo fato de cada núcleo ter conexão com todos os demais, necessitando assim de somente um salto pra mensagem chegar a seu destino. Em contrapartida a topologia com pior desempenho em todos os cenários foi a *FatTreee*. Já a *Torus* foi levemente mais eficiente que a *Mesh* em todos os cenários.

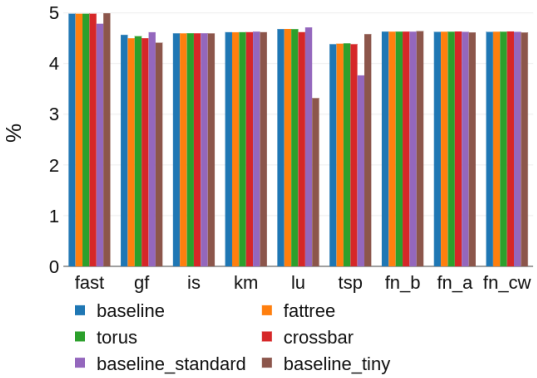
Na maior parte dos casos a variação das configurações dos controladores de memória teve pouco impacto, sem um padrão em comum. De forma análoga o tamanho de links teve pouca influência, porém contendo um padrão de existência de um limitante a partir de *128Kb* em FAST, GF, KM, LU, TSP e FN's, e um comportamento contrário com IS.

Um comportamento interessante é quanto a escalabilidade. Quando com escala 2x4 a o número de saltos diminui em média 30% em relação ao *baseline* (4x4). Entretanto quando com 4x8 o número de saltos aumenta em média 40% em relação ao *baseline*.

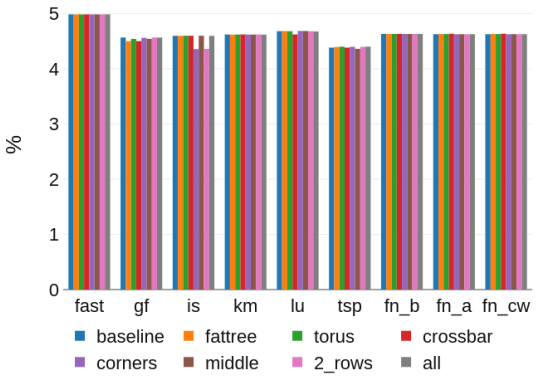
4.4 TAXA DE OCUPAÇÃO MÉDIA DAS FILAS

Foi avaliado o quanto da capacidade de uma fila é utilizada em termos médios. Para calcular tal número foi utilizado o modelo *M/M/1* da teoria das filas. Tal modelo considera que a taxa de ocupação da fila é dada quociente entre um λ representando a taxa média de chegada de elementos na fila, e um ψ representando o tempo médio de espera na fila. Neste contexto o λ é a taxa de injeção de pacotes na rede e o ψ é a latência média dos pacotes nas filas. Assim obtemos os resultados mostrados na Figura 7.

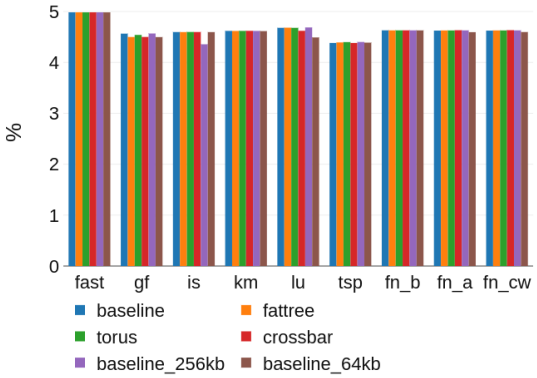
O tamanho na ocupação no tamanho de ocupação das filas no geral foi bem pequeno. O único parâmetro que alterou de forma sensível tal métrica foi os protocolos de cache.



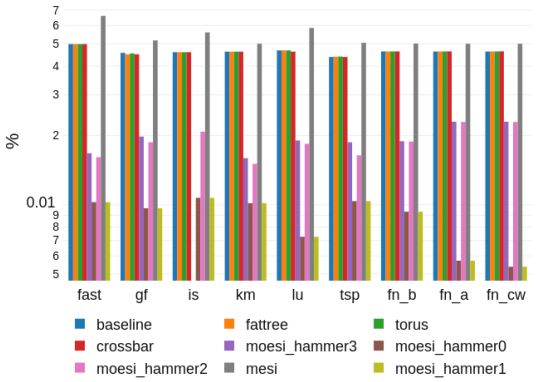
(a) Dataset



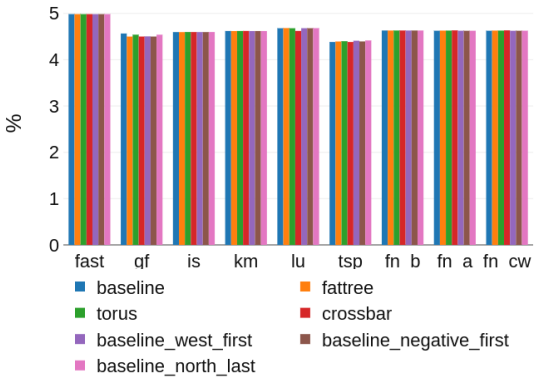
(b) Controlador de memória



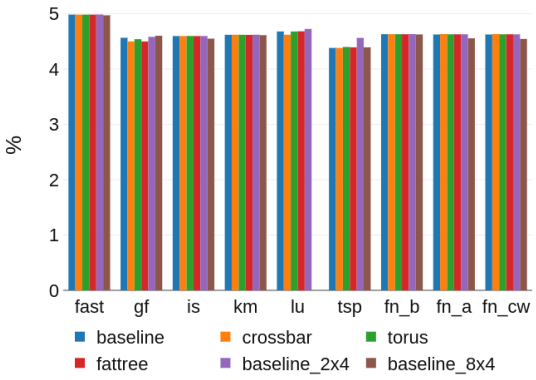
(c) Link



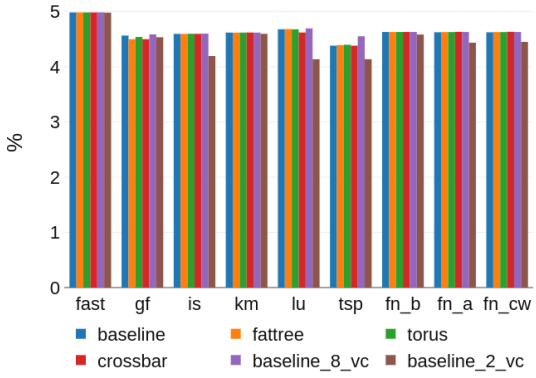
(d) Protocolo



(e) Algoritmo de roteamento



(f) Escalabilidade



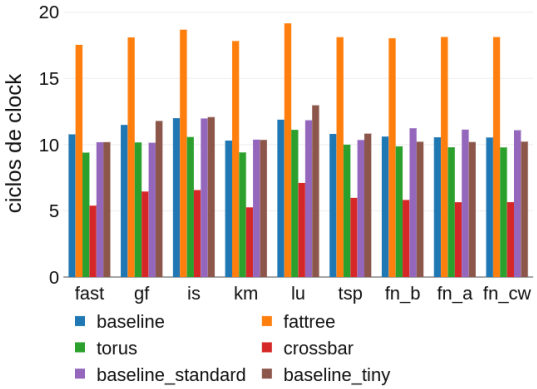
(g) Canais Virtuais

FIGURA 7 – Taxa de ocupação média das filas

Em todos os casos o protocolos MESI foram responsáveis por provocar uma fila maior, e os da família MOESI geraram filas menores.

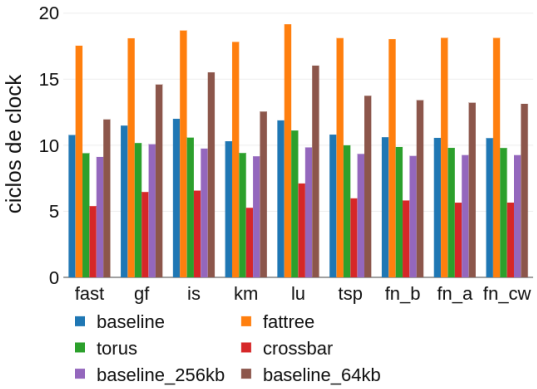
4.5 LATÊNCIA EM CANAIS VIRTUAIS

As topologias *FatTreee* e *Crossbar* foram as responsáveis por provocar, respectivamente, a maior e menor latência média. Entre as topologias *Mesh*, a *all* foi mais eficiênte em todos os cenários, seguido por *2-rows*. Nas aplicações GF e KM o com pior desempenho foi a *middle*, seguido da *corners*, padrão que se altera nas demais. Os resultados podem ser observados na Figura 8.

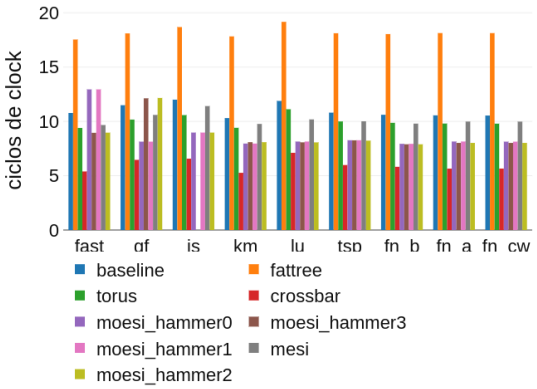


(a) Dataset

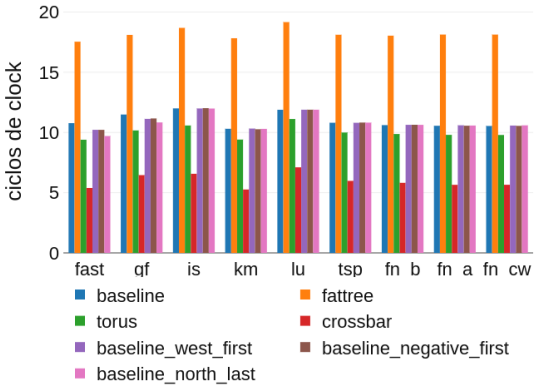
(b) Controlador de memória



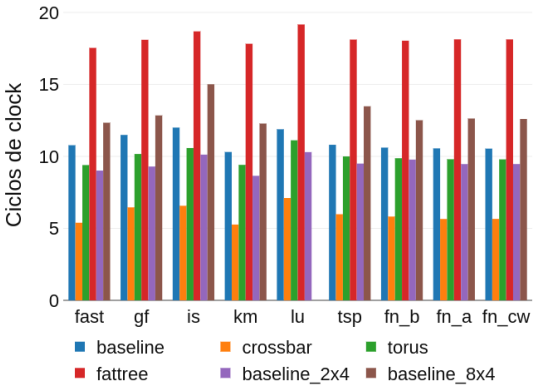
(c) Link



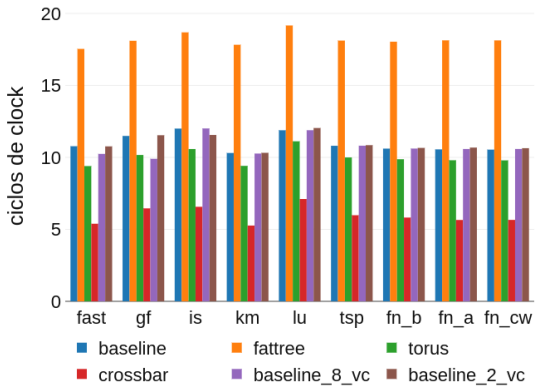
(d) Protocolo



(e) Algoritmo de roteamento



(f) Escalabilidade



(g) Canais Virtuais

FIGURA 8 – Latência média nas Vnet

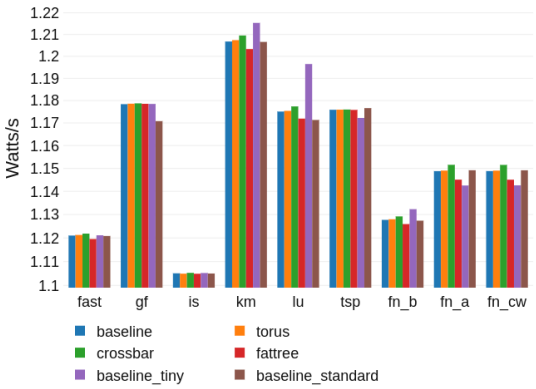
Avaliando a variação de tamanho do *link* é possível afirmar que quanto maior seu tamanho, menor será a latência nos canais virtuais. De forma contrária a esta, quanto menor o tamanho da rede, menor será a latência.

Do ponto de vista de protocolos, os da família MOESI tiveram melhor desempenho frente aos MESI em todos os cenários.

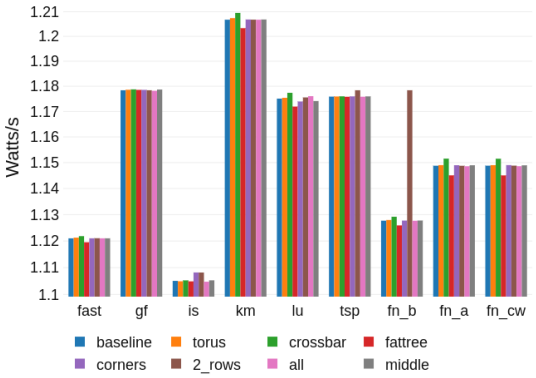
Por fim, o nota-se que o algoritmo de roteamento adotado, bem como a quantidade de canais virtuais praticamente não impactaram no aspecto aqui avaliado.

4.6 ENERGIA CONSUMIDA

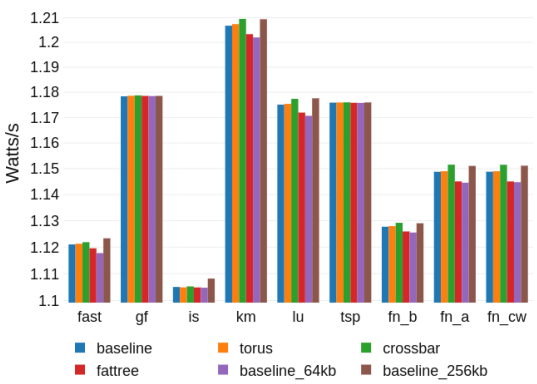
Por fim, o última métrica avaliada é quanto ao uso de energia média por segundo. Para calcular esse dado foi utilizado a ferramenta McPAT (S. LI et al., 2009). Os resultados são mostrados na Figura 9.



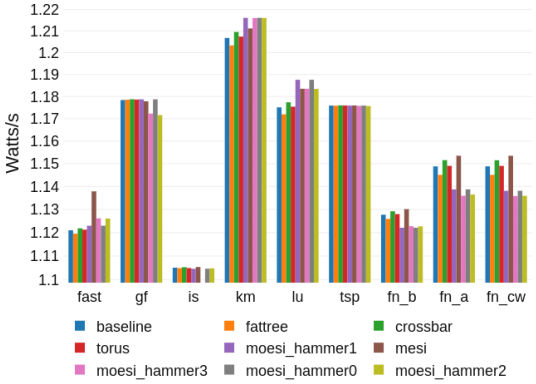
(a) Dataset



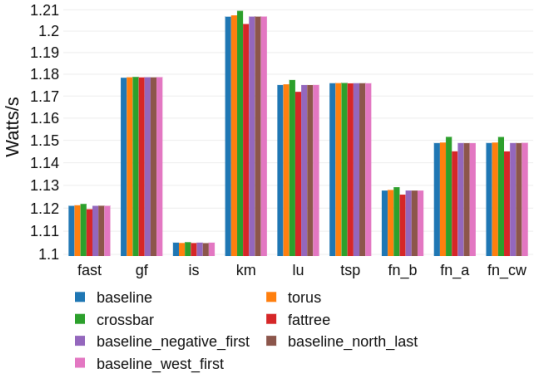
(b) Controlador de memória



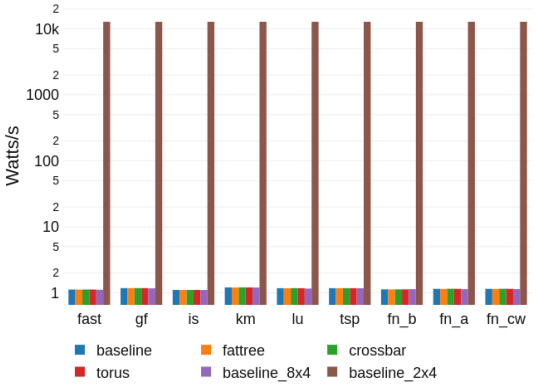
(c) Link



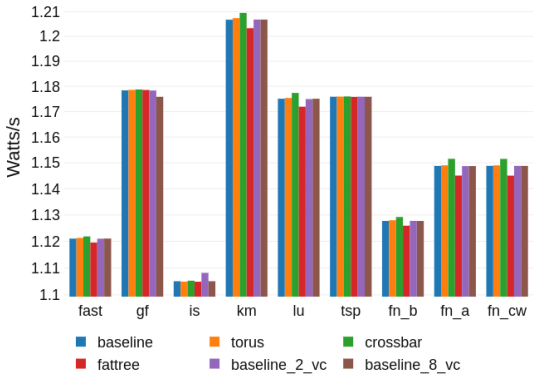
(d) Protocolo



(e) Algoritmo de roteamento



(f) Escalabilidade



(g) Canais Virtuais

FIGURA 9 – Energia Média por segundo

O gasto de energia médio por segundo teve uma variação muito pequena na maioria dos cenários, na escala de casas decimais. Nestes casos, destaque negativo para a disposição *2-rows* com o FN-B. O impacto mais visível ocorreu variando a quantidade de processadores na rede. Quando diminuída para uma escala 2x4, o consumo de energia aumentou mais de 1000%, deixando claro que, quanto menos processadores na rede, mais tempo ele terá que trabalhar no máximo de sua capacidade de processamento.

5 TRABALHOS RELACIONADOS

Tedesco et al. (2005) utilizaram uma NoC com topologia *Mesh* 8x8 para avaliar a rede do ponto de vista interno, permitindo a verificação de desempenho de cada canal da rede; e também do ponto de vista externo, onde a rede é enxergada como uma caixa preta e somente os dados gerados nas interfaces de comunicação externa são avaliados. Para seus experimentos a rede foi modelada em RTL VHDL (MURUGESAN; RANJITHKUMAR, 2010). Os autores avaliaram diferentes algoritmos de roteamento e também diferentes quantidades de canais virtuais existentes nos roteadores.

Holsmark e Kumar (2005) estudaram o conceito de regiões em redes *mesh* em chip. Região segundo eles é um conceito criado para lidar com unidades de processamento que possuem um tamanho maior que os *tiles* da rede. A existência de regiões faz com que algoritmos de roteamento tenham que ser modificados a fim considerar esses pontos na rede, pois seu posicionamento acaba influenciando o desempenho da mesma.

Já com o intuito de auxiliar o controle de NoCs, Clermidy et al. (2009) propõem o uso de um controlador de comunicação e configuração que interage tanto com as unidades de processamento quanto com a interface de rede, tornando mais fácil o gerenciamento e configuração de fluxo e sincronização de dados na rede.

Já Q. Yang e Wu (2010) propuseram uma topologia derivada da *Mesh*, chamada por ele de T-Mesh, que consiste numa *Mesh* onde cada roteador no canto da rede é conectado a outros roteadores adjacentes do canto da rede, formando um anel envolvendo a NoC. Eles também desenvolvem um algoritmo de roteamento, o TXY para dar suporte a essa nova topologia. Por fim, eles avaliaram sua proposta com uma topologia *mesh* tradicional, simulando ambas no simulador *gpNoCsim* (HOSSAIN et al., 2007).

Hao et al. (2011) também fizeram um estudo do impacto de diferentes algoritmos de roteamento em redes *Mesh*, utilizando o simulador NIRGAM (KHAN et al., 2018), porém com foco em analisar os dados de forma separada, isto é, entre os eixos da rede.

Chen, Gillard e C. Li (2012) realizaram um estudo de NoC's sob diferentes configurações a fim de avaliar os *delays* nó-a-nó, largura de banda e taxa de pacotes perdidos na rede. Nesse trabalho foi utilizado o simulador *NS-2* (BUKASHKIN; BURANOVA; SAPRYKIN, 2016). As topologias escolhidas foram *Torus, Metacubo* e *Hipercubo* variando modelos de tráfego e quantidade de nós na rede, concluindo que Torus é a escolha mais viável entre essas para redes com 32 a 64 nós.

Chang e Chiu (2012) focaram seu estudo em mecanismos de comutação dos roteadores em rede em topologias *Mesh* e *H-Star* (J. KIM et al., 2010), analisando os padrões iSLIP (MCKEOWN, 1999) e BvN (Birkhoff-von Neumann) (YE et al., 2017).

Saini e Ahmed (2015) realizaram comparação de desempenho de duas redes *Mesh*, uma 3D e outra hexagonal, ambas com números semelhantes de nós, visando observar a diferença de algoritmos de roteamento 2D e 3D utilizando o simulador *NOXIM* (CATANIA et al., 2015). Os autores concluíram que para taxas baixas de injeção de pacotes, a topologia 2D hexagonal é mais eficiente, tanto do ponto de vista de roteamento quando do ponto de vista energético.

Psathakis et al. (2015) também analisaram NoCs com topologia *Mesh* variando de forma exaustiva parâmetros como concentração de núcleos, tamanho dos *flits* dentre outros, utilizando *energy-throughput ratio* como parâmetro de comparação de desempenho de cada variação proposta.

Gardea et al. (2017) avaliaram por meio do simulador de NoCs *BookSim* (D. U. BECKER et al., 2013) uma rede *Mesh* 3D utilizando os padrões de tráfego *uniform random traffic* e *restricted random traffic* a fim de investigar saturações na rede.

6 CONCLUSÃO

Redes em chip são uma alternativa frente aos métodos de interconexão clássicos adotados para a interligação entre núcleos de processamento. Entretanto, como comentado, seu projeto deve levar em conta diversos fatores. Neste trabalho percebeu-se que o protocolo de coerência de *cache* foi o parâmetro que mais influenciou nos cenários de mesma topologia na rede. De forma contrária, concluiu-se que a variação de quantidade de canais virtuais foi a questão que menos causou impacto na rede. Já entre as topologias notou-se predominância diferentes conforme o quesito avaliado, sugerindo um possível *trade-off* no momento de se projetar uma NoC.

Com as versões próprias do FN observou-se que o balanceamento de carga influencia consideravelmente na vazão da rede e no tráfego médio nos controladores. A versão *FN-B* que foi adotado um caráter de balanceamento por bloco apresentou uma menor vazão na rede, que foi uma desvantagem frente as suas outras versões, porém ela superou as demais versões pelo fato de exigir menos o uso dos controladores de memória no quesito de tráfego médio através deles.

Foi observado também em certos cenários como por exemplo o do IS em que o tamanho do link maior acarreta em uma vazão menor na rede, dentre outros exemplos de aplicações que dentro de um quesito avaliado possuem comportamento completamente diferente da maioria, sugerindo que talvez exista uma influência do modelo de paralelização adotado pelas aplicações impactando no desempenho arquitetural da NoC. Portanto fica aqui tal questão como sugestão de trabalhos futuros investigar se modelos como divisão e conquista, *map*, *map reduce*, *stencil* e *workpool* impactam de forma considerável no desempenho de uma NoC em aspectos análogos aos estudados nesse trabalho.

Com esta pesquisa espera-se contribuir com a comunidade científica mostrando, através de simulação *full-system* com as aplicações utilizadas, que certas questões de projeto que parecem óbvias que irão melhorar o desempenho da rede, como aumentar o número de canais virtuais, aumentar sua escala, aumento do tamanho do *link* dentre outras, nem sempre acabam retornando esse resultado esperado, e também de certa forma em alguns casos podem até causar um resultado completamente contrário do esperado.

REFERÊNCIAS

AGARWAL, Niket et al. GARNET: A detailed on-chip network model inside a full-system simulator. In: IEEE. PERFORMANCE Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on. 2009. p. 33–42.

BO WANG et al. The congestion controlling algorithm based on dynamic routing table on network on chips. In: 10TH International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM 2014). Setembro 2014. p. 381–385.

BORKAR, Shekhar. Getting Gigascale Chips: Challenges and Opportunities in Continuing Moore's Law. **Queue**, ACM, New York, NY, USA, v. 1, n. 7, 2003.

BRYANT, Randal; BECKMANN, Nathan. **Directory-Based Cache Coherence II + Memory Consistency Models**. 2012. Disponível em: <https://www.cs.cmu.edu/afs/cs/academic/class/15418-s12/www/lectures/13_directorycoherence2.pdf>.

BUKASHKIN, S.; BURANOVA, M.; SAPRYKIN, A. An analysis of multimedia traffic in the MPLS network in ns2 simulator. In: 2016 Third International Scientific-Practical Conference Problems of Infocommunications Science and Technology (PIC S T). Outubro 2016. p. 185–188.

BUTKO, A. et al. Accuracy evaluation of GEM5 simulator system. In: 7TH International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC). Jul. 2012. p. 1–7.

CATANIA, V. et al. Noxim: An open, extensible and cycle-accurate network on chip simulator. In: 2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP). Jul. 2015. p. 162–163.

CHANG, Y.; CHIU, C. A Study of NoC Topologies and Switching Arbitration Mechanisms. In: 2012 IEEE 14th International Conference on High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems. Jun. 2012. p. 1643–1647.

CHEN, J.; GILLARD, P.; LI, C. Performance evaluation of three Network-on-Chip (NoC) architectures (Invited). In: 2012 1st IEEE International Conference on Communications in China (ICCC). Agosto 2012. p. 91–96.

CLERMIDY, F. et al. A Communication and configuration controller for NoC based reconfigurable data flow architecture. In: 2009 3rd ACM/IEEE International Symposium on Networks-on-Chip. Maio 2009. p. 153–162.

CONWAY, P. et al. Cache Hierarchy and Memory Subsystem of the AMD Opteron Processor. **IEEE Micro**, v. 30, n. 2, p. 16–29, mar. 2010.

COTA, Érika; MORAIS AMORY, Alexandre de; LUBASZEWSKI, Marcelo Soares. **Reliability, Availability and Serviceability of Networks-on-Chip**. Springer, 2011.

D. U. BECKER, and et al. A detailed and flexible cycle-accurate Network-on-Chip simulator. In: 2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). Abr. 2013. p. 86–96.

ENGBLOM, Jakob. Full-System Simulation, fev. 2019.

GARDEA, Jesus et al. Performance Evaluation of Mesh-based 3D NoCs. In: PROCEEDINGS of the 10th International Workshop on Network on Chip Architectures. Cambridge, MA, USA: ACM, 2017. (NoCArc'17), 7:1–7:6.

HAO, Pan et al. Comparison of 2D MESH routing algorithm in NOC. In: 2011 9th IEEE International Conference on ASIC. Outubro 2011. p. 791–795.

HENNESSY, John L. **Arquitetura de Computadores**. São Paulo: Elsevier, 2013.

HOLSMARK, R.; KUMAR, S. Design issues and performance evaluation of mesh NoC with regions. In: 2005 NORCHIP. Nov. 2005. p. 40–43.

HOSSAIN, H. et al. Gpnocsim - A General Purpose Simulator for Network-On-Chip. In: 2007 International Conference on Information and Communication Technology. Mar. 2007. p. 254–257.

KHAN, S. et al. Comparative analysis of network-on-chip simulation tools. **IET Computers Digital Techniques**, v. 12, n. 1, p. 30–38, 2018.

KIM, H. J.; SEO, J. T.; HAN, T. H. 3CEO: Three dimensional Cmesh based electrical-optical router for networks-on-chip. In: ICTC 2011. Set. 2011. p. 114–119.

KIM, J. et al. A 118.4 GB/s Multi-Casting Network-on-Chip With Hierarchical Star-Ring Combined Topology for Real-Time Object Recognition. **IEEE Journal of Solid-State Circuits**, v. 45, n. 7, p. 1399–1409, jul. 2010.

LI, S. et al. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In: 2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). Dezembro 2009. p. 469–480.

MA, Sheng et al. **Networks-on-Chip: From Implementations to Programming Paradigms**. Morgan Kaufmann, 2014.

MCKEOWN, N. The iSLIP scheduling algorithm for input-queued switches. **IEEE/ACM Transactions on Networking**, v. 7, n. 2, p. 188–201, abr. 1999.

MURUGESAN, S.; RANJITHKUMAR, P. Satisfiability based test generation for stuck-at fault coverage in RTL circuits using VHDL. In: 2010 Second International conference on Computing, Communication and Networking Technologies. Jul. 2010. p. 1–6.

NYCHIS, George et al. Next Generation On-chip Networks: What Kind of Congestion Control Do We Need? In: PROCEEDINGS of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. Monterey, California: ACM, 2010. (Hotnets-IX), 12:1–12:6.

PALESI, Maurizio; DANESHTALAB, Masoud (Ed.). **Routing Algorithms in Networks-on-Chip**. New York: Springer, 2014.

PATTERSON, David A.; HENNESSY, John L. **Computer Organization and Design: The Hardware/Software Interface (The Morgan Kaufmann Series in Computer Architecture and Design)**. São Paulo: Elsevier, 2008.

PSATHAKIS, A. et al. A systematic evaluation of emerging mesh-like CMP NoCs. In: 2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS). Maio 2015. p. 159–170.

QAWASMEH, A.; MALIK, A. M.; CHAPMAN, B. M. OpenMP Task Scheduling Analysis via OpenMP Runtime API and Tool Visualization. In: 2014 IEEE International Parallel Distributed Processing Symposium Workshops. Maio 2014. p. 1049–1058.

ROS, Alberto; E., Manuel; M., Jose. Cache Coherence Protocols for Many-Core CMPs. In: PARALLEL and Distributed Computing. InTech, jan. 2010.

SAINI, Ravindra; AHMED, Mushtaq. 2D Hexagonal Mesh Vs 3D Mesh Network on Chip: A Performance Evaluation. In: v. 4, p. 2210–142.

SANCHEZ, Daniel; MICHELOGIANNAKIS, George; KOZYRAKIS, Christos. An Analysis of On-chip Interconnection Networks for Large-scale Chip Multiprocessors. **ACM Trans. Archit. Code Optim.**, ACM, New York, NY, USA, v. 7, n. 1, 4:1–4:28, maio 2010.

SEELY, S. Monitor preprocessor for Pthreads. In: 34TH Annual Frontiers in Education, 2004. FIE 2004. Outubro 2004. s1h/23–s1h/27 vol. 3.

SOUZA, Matheus A. et al. CAP Bench: a benchmark suite for performance and energy evaluation of low-power many-core processors. **Concurrency and Computation: Practice and Experience**, v. 29, n. 4, 2017.

STENSTRÖM, Per; BRORSSON, Mats; SANDBERG, Lars. An adaptive cache coherence protocol optimized for migratory sharing. In: ISCA '93. 1993.

TEDESCO, L. et al. Traffic Generation and Performance Evaluation for Mesh-based NoCs. In: 2005 18th Symposium on Integrated Circuits and Systems Design. Set. 2005. p. 184–189.

AL-WAISI, Z.; AGYEMAN, M. O. An overview of on-chip cache coherence protocols. In: 2017 Intelligent Systems Conference (IntelliSys). Set. 2017. p. 304–309.

XU, T. C.; LILJEBERG, P.; TENHUNEN, H. Optimal memory controller placement for chip multiprocessor. In: 2011 Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS). Outubro 2011. p. 217–226.

YANG, J. et al. Adding TBB Contents to the Multi-Core Related Curriculums. In: 2008 First International Conference on Intelligent Networks and Intelligent Systems. Nov. 2008. p. 685–688.

YANG, Q.; WU, Z. An Improved Mesh Topology and Its Routing Algorithm for NoC. In: 2010 International Conference on Computational Intelligence and Software Engineering. Dez. 2010. p. 1–4.

YE, T. et al. Deflection-Compensated Birkhoff–von-Neumann Switches. **IEEE/ACM Transactions on Networking**, v. 25, n. 2, p. 879–895, abr. 2017.

ZAMANIFAR, K.; NEMATBAKHSH, N.; SADJADY, R. S. A New Load Balancing Algorithm in Parallel Computing. In: 2010 Second International Conference on Communication Software and Networks. Fevereiro 2010. p. 449–453.

APÊNDICES

APÊNDICE A – FRIENDLY NUMBERS

Quando se trabalha com computação paralela, uma das principais questões que surgem é como distribuir as tarefas e dados entre as unidades de processamento. No caso ideal, todas as unidades devem realizar a mesma quantidade de trabalho. Essa distribuição pode ser realizada estaticamente, isto é, em tempo de compilação, ou dinamicamente, que é um tratamento mais complexo que dá a capacidade de em tempo de execução distribuir as tarefas conforme obtém informações de uso das unidades de processamento Zamanifar, Nematbakhsh e Sadjady (2010).

Neste contexto, ao avaliar o comportamento das aplicações do benchmark CAP, concernente a escalabilidade, observamos que a aplicação FN não possui um balanceamento de carga ideal. Desta forma desenvolvemos duas novas versões para essa aplicação.

A.1 A APLICAÇÃO FN

O algoritmo em questão implementado é o FN. Ele foi construído de forma modular em linguagem C++, permitindo a fácil variação via parâmetros de ferramentas de paralelização e formas de balanceamento de carga.

As ferramentas de paralelização utilizadas foram *Posix PThread*(SEELY, 2004), *OpenMP* (QAWASMEH; MALIK; CHAPMAN, 2014) e *Intel TBB* (J. YANG et al., 2008), que são ferramentas bastante populares e oferecem um paralelismo a nível de *threads*.

Também foram implementadas duas novas técnicas de balanceamento de carga focadas em distribuição dos dados, já que o algoritmo consiste em achar pares de números amigos em um intervalo estabelecido. Tais técnicas tem o caráter estático pois seu comportamento é pré-definido e não se altera conforme o programa é executado. As novas técnicas implementadas foram nomeadas da seguinte maneira:

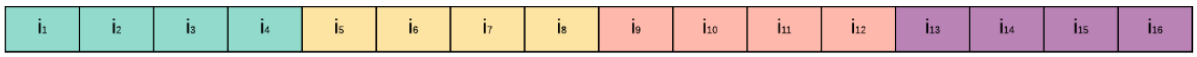
Balanceamento alternante: os dados do conjunto são distribuídos de forma alternada entre as unidades de processamento. Cada unidade toma como ponto de partida no conjunto de dados o índice correspondente ao seu *id*, e, os próximos dados são obtidos por meio de saltos, do tamanho de unidades existentes, a partir da posição corrente.

Balanceamento de peso constante: essa é uma técnica mais elaborada que leva em conta a grandeza dos números no conjunto de dados. Como o *FN* é um problema matemático cujo uma das etapas na implementação consiste em realizar a soma dos divisores de um número, essa técnica tenta minimizar a diferença de

operações de divisão para cada unidade de processamento, distribuindo entre as unidades números de forma que a soma dos números que cada unidade trabalha seja igual.

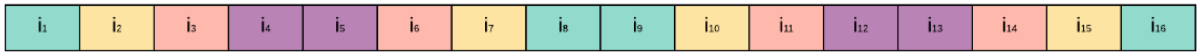
Por padrão, o algoritmo do *benchmark* CAP adota uma estratégia de dividir igualmente o intervalo de entrada para cada unidade de processamento. Sendo assim, nosso algoritmo também permite esse tipo de comportamento, onde aqui o chamaremos de *balenceamento de bloco*.

A Figura 7 ilustra como os balanceamentos acontecem. Cada cor representa o dado de uma unidade de processamento.



(a) Balanceamento de Bloco.

(b) Balanceamento Alternante.



(c) Balanceamento de Peso Constante.

FIGURA 10 – Técnicas de balanceamento

Vamos ver matematicamente a quantidade de operações de divisão que cada unidade tem que realizar com cada estratégia de balanceamento. Suponha que o conjunto de entrada inicia em 0 e termina em 11 e também que haja 4 unidades, conforme ilustrado na Figura anterior e que, para achar todos os divisores de um número, com uma implementação não otimizada, a quantidade de operações de divisão seja igual ao valor do número. Assim, a quantidade de operações realizadas por cada técnica é como segue:

Balanceamento de bloco

$P_{verde} = 0 + 1 + 2 + 3 = 6$ operações
$P_{amarelo} = 4 + 5 + 6 + 7 = 22$ operações
$P_{vermelho} = 8 + 9 + 10 + 11 = 38$ operações
$P_{roxo} = 12 + 13 + 14 + 15 = 54$ operações

Balanceamento alternante

$P_{verde} = 0 + 4 + 8 + 12 = 24$ operações
$P_{amarelo} = 1 + 5 + 9 + 13 = 28$ operações
$P_{vermelho} = 2 + 6 + 10 + 14 = 32$ operações
$P_{roxo} = 3 + 7 + 11 + 15 = 36$ operações

Balanceamento de peso constante
$P_{verde} = 0 + 7 + 8 + 15 = 30$ operações
$P_{amarelo} = 1 + 6 + 9 + 14 = 30$ operações
$P_{vermelho} = 2 + 5 + 10 + 13 = 30$ operações
$P_{roxo} = 3 + 4 + 11 + 12 = 30$ operações

A.2 OTIMIZAÇÕES

Algumas soluções para otimizar o algoritmo e torná-lo *thread-safe* foram elaboradas. São elas:

Calculadora: cada unidade de processamento vai possuir um objeto cujo a função é achar divisores de um número e, seus respectivos pares.

Geradores de Índice: cada calculadora possui um gerador de índice próprio que, baseado no *id* da unidade de processamento, tamanho do intervalo de entrada e posição corrente, decide qual é índice do conjunto sob o qual a calculadora terá que realizar suas operações.

Já visando otimizar as computações, são adotadas as seguintes abordagens:

Limite de iteração: A tarefa de procurar divisores de um número pode ter sua iteração limitada superiormente não pelo valor do número envolvido, mas sim pela sua raiz quadrada.

Cache: Foi criada uma estrutura de cache para armazenar a soma dos divisores de cada número, evitando re-cálculo.

A.3 CONFIGURAÇÃO DOS EXPERIMENTOS

Diferentemente dos experimentos anteriores, nestes não será utilizado um simulador, e sim um computador real. As configurações do ambiente de simulação é exemplificado na 5 abaixo:

Processador	2x Intel Xeon E5-2630
SO	Smubuntu
Compilador	GCC
Threads	2-24

TABELA 5 – Configuração do sistema

Cada versão do algoritmo foi executada 10 vezes visando a obtenção de uma média aritmética dos dados onde esses foram coletados utilizando as ferramentas Perf e Pin da Intel.

A.4 RESULTADOS

Primeiramente apresentamos na Figura 11 o resultado de *speedup* para cada versão. Conforme esperado, o balanceamento por bloco para todas as versões obteve um desempenho pior devido não considerar durante a distribuição a grandeza dos números, que é uma questão importante para esse algoritmo. Porém o balanceamento alternante e o de peso constante mostraram resultados muito parecidos. Isso ocorre devido a otimização de realizar cálculo dos divisores até a raiz quadrada do número somente, o que acaba minimizando a diferença de operações matemáticas realizadas entre os dois modos, fazendo com que questões como o escalonamento de tarefas do processador por exemplo acabem tendo maior influência no desempenho destes.

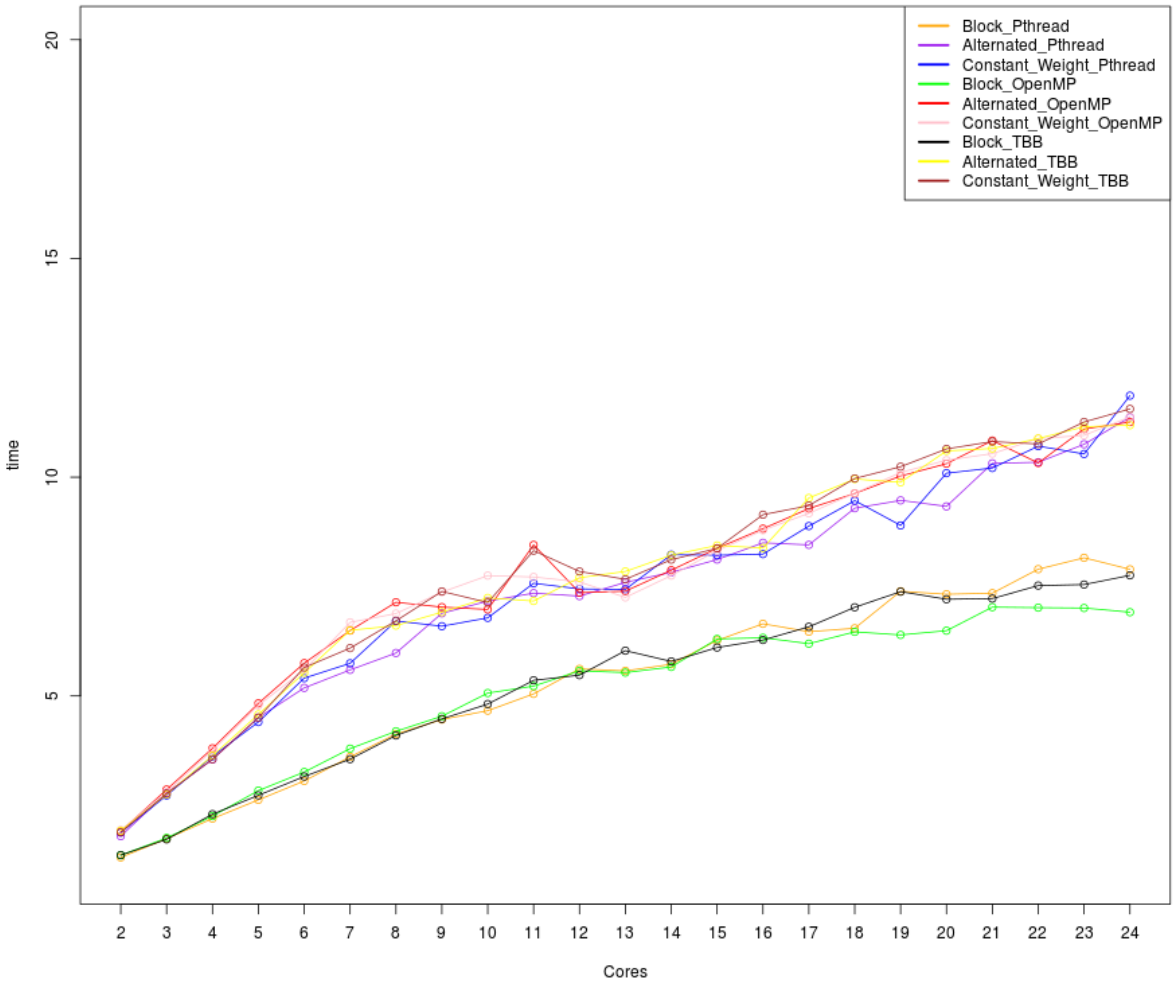


FIGURA 11 – Speedup

Já quando analisamos os dados de instruções por ciclo (IPC) notamos que a ferramenta *TBB* mostrou um comportamento mais instável conforme mostrado na Figura 12. Entretanto a grande maioria quando próximo a 12 *threads* acabam obtendo um uma estabilidade na sua progressão.

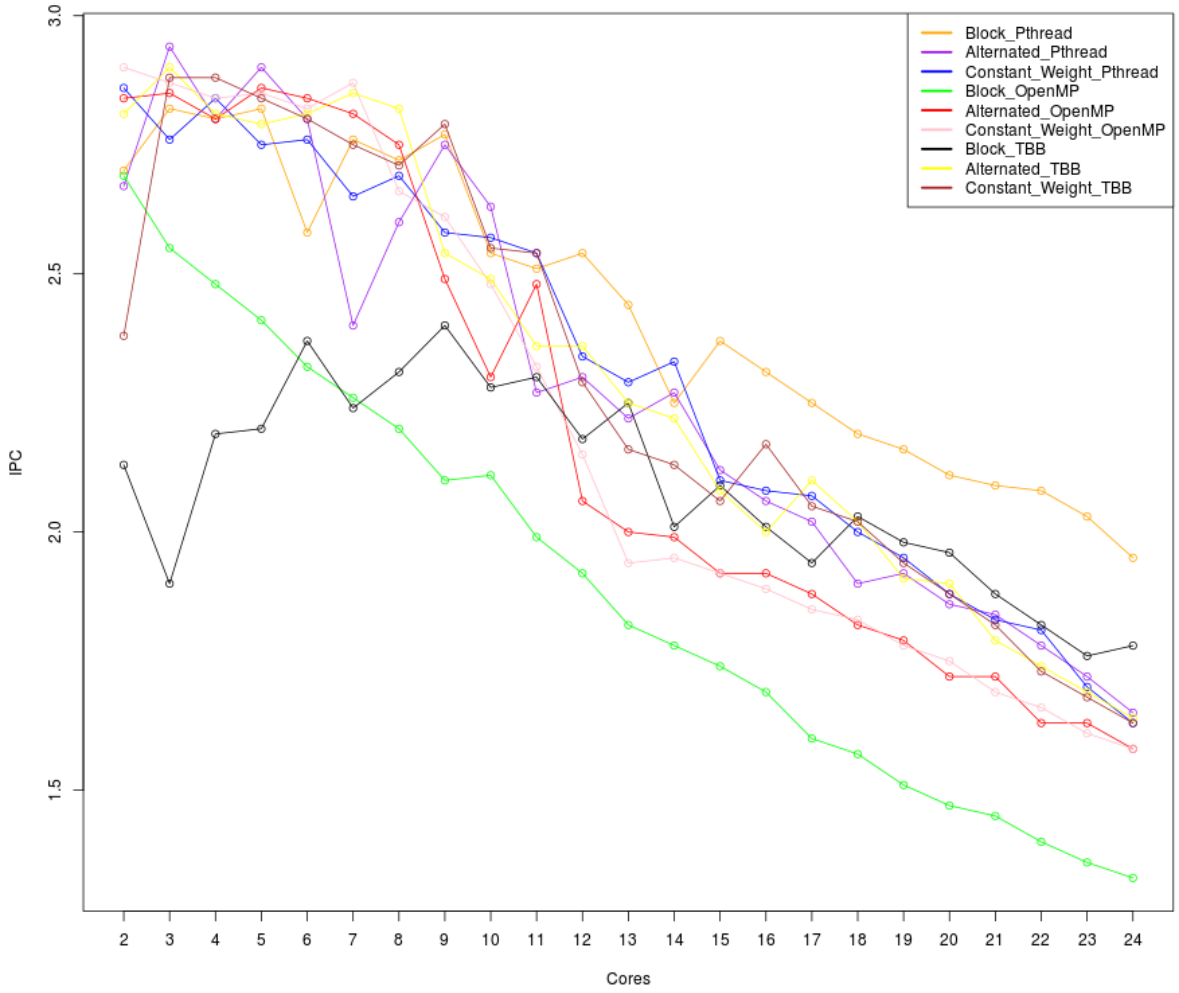


FIGURA 12 – IPC

O balanceamento por bloco também foi o que provocou o maior número de *cache-misses* conforme mostrado na Figura 13. Isso também é um fator que fez com que esse balanceamento tivesse um *speedup* inferior aos demais, pois quanto mais falhas na cache mais acessos a memória principal o processador teve que realizar.

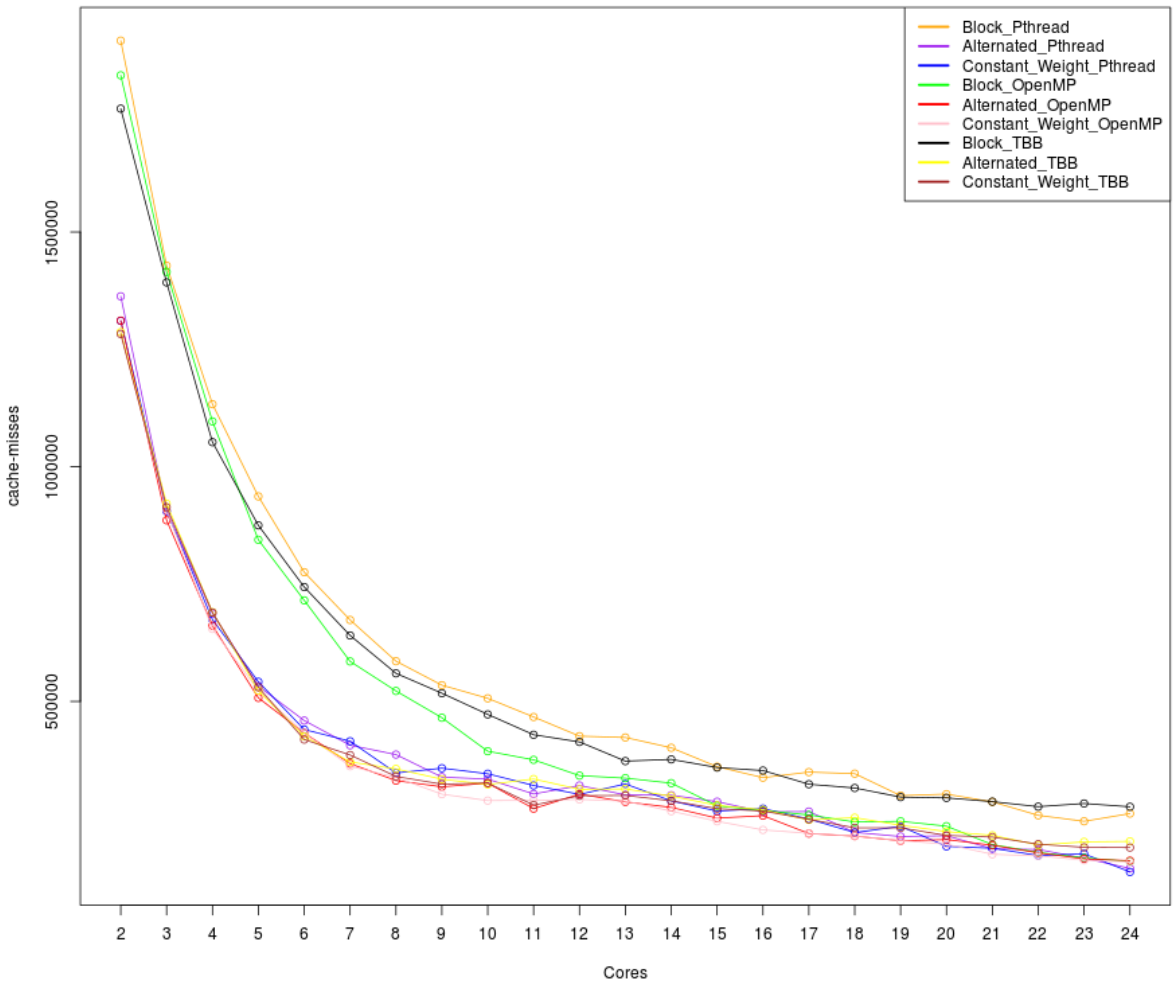


FIGURA 13 – Cache Misses

Falhas de páginas foi outra questão que observamos nos experimentos (Figura 14). Neste quesito a ferramenta *Intel TBB* mostrou-se pior que as outras para todas as versões, provocando uma quantidade de falhas em um grau de evolução maior que as outras conforme o número de *threads* aumenta.

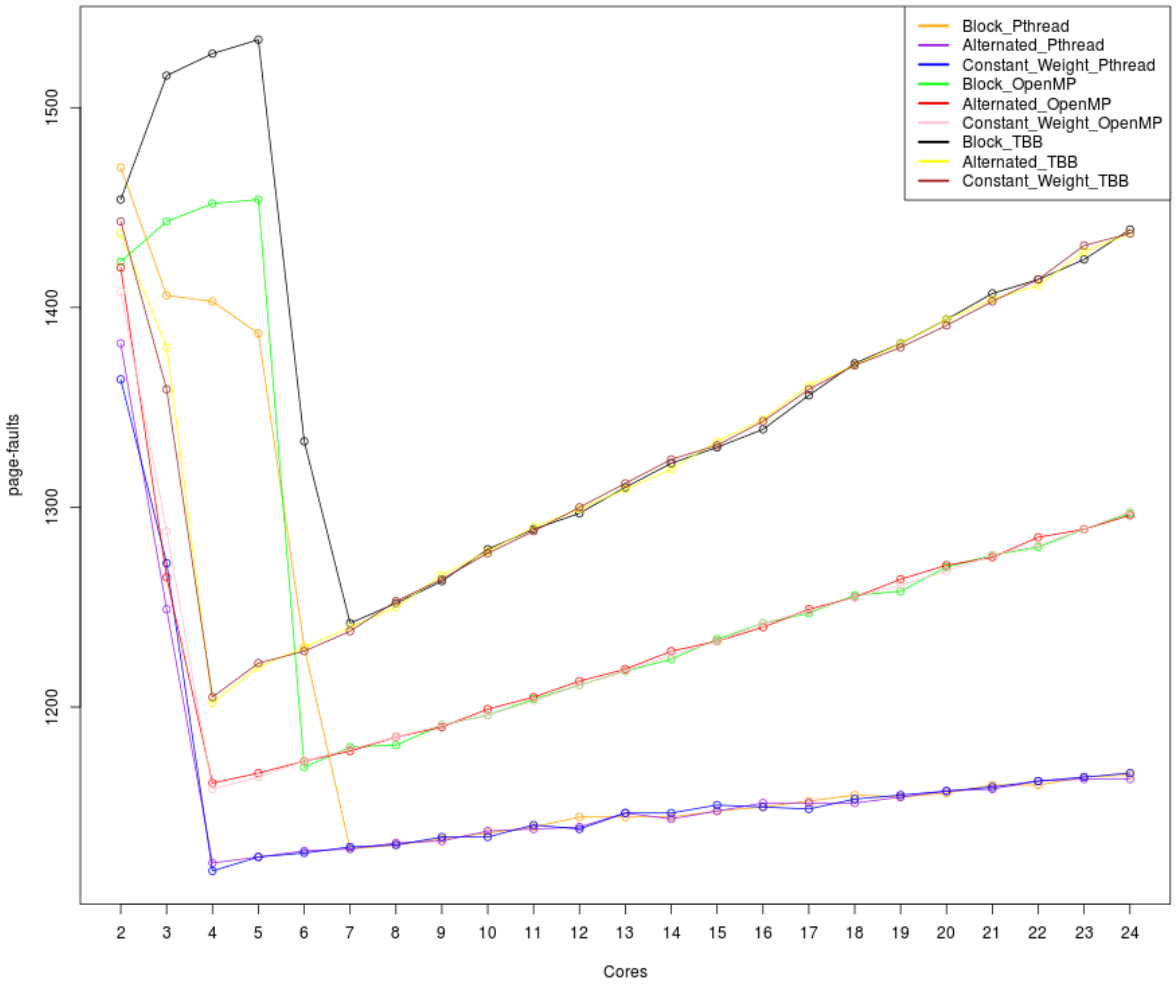


FIGURA 14 – Page Faults

E por fim o ultimo parâmetro analisado foi a questão de trocas de contexto. Conforme apresentado na Figura 15, as versões que utilizam *pthreads* começam a ter um número de troca de contextos considerável quando próximos do número de núcleos físicos do processador, que é 12. Isso ocorre de forma mais sucinta nas outras estratégias, entretanto sempre com destaque quando em conjunto com o balanceamento por bloco.

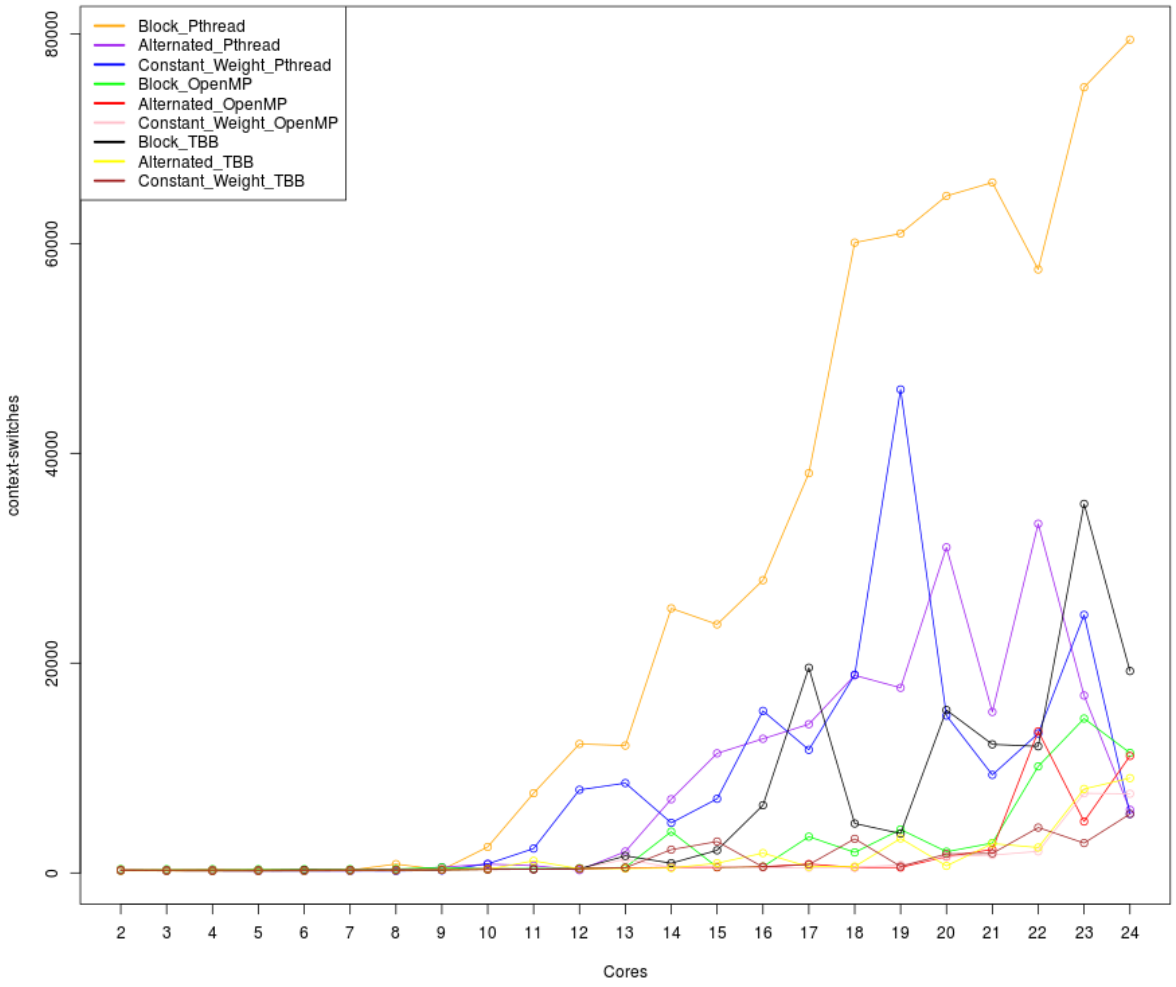


FIGURA 15 – Context Switches

Os experimentos indicam que o balanceamento por bloco é o menos eficiente, *OpenMP* combinado com o balanceamento alternante ou de peso constante é a melhor escolha.