Simulação de sistemas *multicore* para fins didáticos

Isabelle Ferreira Silva Nunes[1], **Liana Duenha**[1]

[1]Faculdade de Computação – Universidade Federal de Mato Grosso do Sul (UFMS)
Cidade Universitária – Campo Grande – MS – Brasil

bellefsnunes@gmail.com, lianaduenha@facom.ufms.br

Abstract. *MPSoCBench is a framework to simulate multicore platforms used in graduate and undergraduate researches for the development and validation of new features. The goal of this paper is to present a friendly Graphical User Interface (GUI) to facilitate the MPSoCBench install and use, and also the interpretation of results. It is expected to expand the target audience and the use cases of MPSoCBench, allowing the tool to become an option to assist and complement the methodology adopted in disciplines such as Computer Architecture, Compilers, Assembly Language, Parallel Programming and Operating Systems.*

Resumo. *O MPSoCBench é um framework para simular plataformas multicore utilizada em pesquisas de iniciação científica e projetos em nível de pós-graduação para o desenvolvimento e validação de novas funcionalidades. O objetivo deste artigo é fornecer uma Interface Gráfica de Usuário (GUI) amigável para facilitar instalação e uso do MPSoCBench e interpretação dos resultados. Espera-se ampliar o público-alvo de usuários e os casos de uso do MPSoCBench, permitindo que a ferramenta se torne uma opção para auxiliar no ensino e complementar a metodologia adotada em disciplinas como Arquitetura de Computadores, Compiladores, Linguagem de Montagem, Programação Paralelo e Sistemas Operacionais.*

1. Introdução

A disciplina Arquitetura de Computadores é comumente ministrada em uma abordagem teórica, buscando a formação conceitual sobre tópicos relevantes do conjunto de instruções que dão base ao projeto de processadores, interface hardware/software, análise de desempenho e a evolução para sistemas com múltiplos processadores (*multicore*). Contudo, torna-se cada vez mais necessário o uso de ferramentas para auxiliar no processo de ensino e aprendizagem, tornando essencial a aplicação de metodologias que possibilitem a conexão entre a teoria e a prática, embasando o aprendizado por meio de experimentação, interpretação e análise de resultados.

O trabalho apresentado em [Duenha and Azevedo 2016] apresenta uma metodologia de ensino complementar às disciplinas de Arquitetura de Computadores e correlatas, principalmente àquelas que utilizam projetos de arquiteturas RISC. Especificamente, o trabalho descreve os resultados obtidos em uma disciplina de Tópicos em Arquitetura de Computadores em que utilizou-se do MPSoCBench [Duenha et al. 2014, Duenha et al. 2016], entre outras ferramentas de simulação, para o apoio pedagógico. Em sequência, o MPSoCBench também foi utilizado na disciplina de Arquitetura de Computadores na graduação e no nível de mestrado [Duenha et al. 2017]. Em suma, foi obser-

vado um bom desempenho no aprendizado dos alunos com o uso da *framework*, entretanto, o processo de instalação e adaptação com a ferramenta, assim como a manipulação dos dados resultantes, levaram mais tempo do que o esperado.

O MPSoCBench é um *framework* para simular plataformas *multicore*, que permite configurar, compilar e simular centenas de distintas configurações de plataformas com até 64 processadores, 4 diferentes ISAs, diferentes mecanismos de interconexão e aplicações paralelas [Duenha et al. 2014, Duenha et al. 2016]. Essa ferramenta tem sido utilizada em pesquisas de iniciação científica e projetos em nível de pós-graduação para o desenvolvimento e validação de novas funcionalidades [Santos et al. 2018]. Por ser executada no Linux (em linha de comando) e dependente de várias outras ferramentas como SystemC [sys 2012] e ArchC [Azevedo et al. 2005], o MPSoCBench torna-se mais adequado para alunos e pesquisadores mais avançados; alunos de graduação relatam dificuldade para utilizar a ferramenta e interpretar os dados de saída.

O objetivo deste trabalho de iniciação científica é fornecer ao usuário do MPSoCBench uma Interface Gráfica de Usuário (GUI) que substitua a entrada de dados de configuração dos simuladores originalmente feita por linha de comando e arquivos de texto, e facilite a visualização e interpretação dos dados resultantes da simulação. Com isso, espera-se que haja maior adesão de usuários atuantes na pesquisa na área de sistemas computacionais e no ensino de graduação e pós-graduação, complementando a metodologia adotada em disciplinas como Arquitetura de Computadores, Compiladores, Linguagem de Montagem, Programação Paralela e Sistemas Operacionais [Duenha and Azevedo 2016].

Esse artigo está organizado como segue: a Seção 2 contém alguns conceitos que fundamentaram nossa pesquisa, bem como a descrição da ferramenta MPSoCBench original; a Seção 3 contém a metodologia para desenvolvimento da interface e novas funcionalidades; a Seção 4 mostra alguns exemplos de uso e resultados de experimentos; a Seção 5 conclui esse artigo.

2. O MPSoCBench

SoCs (do inglês, *System-On-Chip*) passam por uma crescente evolução desde sua criação, tendo assim uma ampla utilização em dispositivos que possuem Circuitos Integrados. Uma de suas evoluções são os MPSoCs (do inglês, *Multiprocessor System-on-chip*), que aplica multiprocessadores no mesmo chip. O MPSoCBench [Duenha et al. 2014] é um *framework* de simulação de MPSoCs construídos sobre arquiteturas RISC (do inglês, *Reduced Instruction Set Computer*): ARM, MIPS, PowerPC e SPARC.

A Figura 1 contém o fluxograma para configuração, construção e execução dos simuladores. Um *script* recebe os parâmetros que descrevem as plataformas as quais deseja simular e dispara uma sequência de ações de configuração dos seus componentes. Da esquerda para direita, vemos que podem ser selecionadas plataformas homogêneas utilizando 1 ou mais *cores* de distintos ISAs (MIPS, ARM, PowerPC, SPARC), duas das quais possuem modelos para consumo energético (SPARC e ARM), configurações de caches (capacidade, políticas de escrita e substituição de blocos, etc) e memórias, mecanismos de interconexão (redes de temporização aproximada, *crossbar* ou roteadores). A partir destes dados são gerados um ou mais simuladores de MPSoCs, que serão responsáveis por executar a simulação funcional de um conjunto de aplicações paralelas que

também foram selecionadas pelo usuário e compiladas para a arquitetura-alvo juntamente com uma biblioteca de programação paralela emulada pela acPthread. Após a simulação, há a avaliação de corretude e geração de relatórios de desempenho na tela e em formato textual [Duenha et al. 2016].

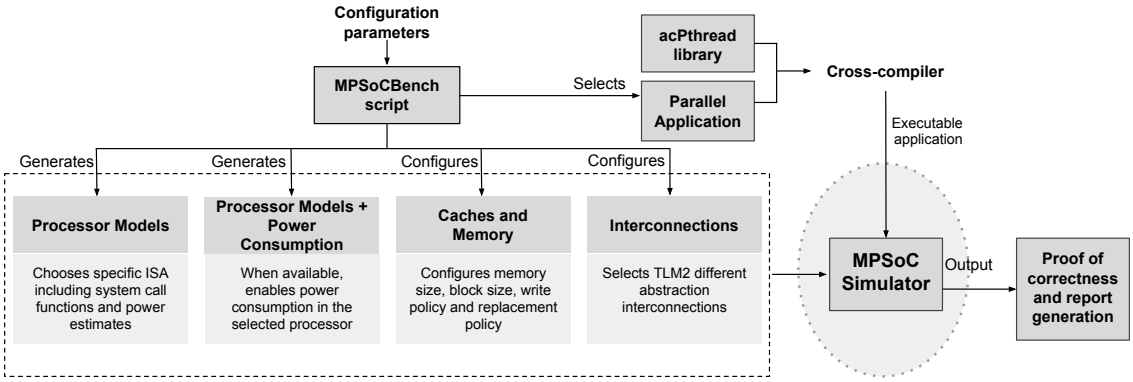

Figura 1. Fluxograma para configuração e execução de simuladores de MPSoCs no MPSoCBench

3. Metodologia

O objetivo desta seção é descrever as funcionalidades implementadas como extensão do MPSoCBench. Basicamente, o principal foco do trabalho é prover uma infraestrutura básica e amigável para configuração das plataformas *multicores* e a exportação de resultados da simulação.

3.1. Desenvolvimento Interface

O *script* original para configuração e criação dos simuladores a partir do MPSoCBench foi desenvolvido em Python 3. Seguindo este mesmo viés, para a elaboração da Interface foi utilizado a mesma versão de Python com importação da biblioteca Tkinter, que já é uma ferramenta GUI padrão da linguagem. As principais vantagens que essa biblioteca proporciona é a simplicidade em manipular os *widgets*, além de possuir tutoriais de fácil acesso [Tkinterbook 2005]. Da mesma forma que o *script* já em funcionamento, a GUI está em inglês.

Nota-se que a GUI pode substituir o *script* do MPSoCBench. De toda forma, suas funcionalidades de construção e execução das plataformas válidas permanecem. O benefício da Interface é torna-se mais simples realizar a configuração básica dos MPSoCs e visualizar os simuladores já criados. Além disso, é possível criar com 5 cliques do *mouse* uma quantidade grande de plataformas distintas (em realidade, centenas de simuladores de MPSoCs). No caso de uma configuração não viável, mensagens são fornecidas ao usuário, sem prejuízo para a construção das demais configurações possíveis.

A Figura 2 ilustra a tela principal com acréscimos de destaques coloridos em 7 distintas subdivisões para detalhá-las no decorrer deste parágafro. A subidivisão marcada como 1 (em vermelho) refere-se à seleção de um ou mais ISAs. Caso o usuário selecione mais de um, serão geradas distintas plataformas homogêneas (e não heterogêneas). A subdivisão marcada como 2 (em roxo) permite que o usuário selecione o mecanismo

de interconexão intrachip: pode-se utilizar redes intrachip (com distintas temporizações baseadas em TLM2) ou um simples *crossbar* (ou Router). A seleção de mais de um mecanismo gera a criação de distintas plataformas, uma para cada mecanismo de interconexão. A subdivisão marcada com 3 (em azul) refere-se à quantidade de núcleos da plataforma, podendo variar entre 1, 2, 4, 8, 16, 32 e 64 *cores*. A subdivisão 4 (em rosa) refere-se a quais aplicações devem ser preparadas para executar nas plataformas configuradas até então. Algumas aplicações necessitam de uma quantidade mínima ou tem um limite máximo de *cores* para que funcionem corretamente (limitação do próprio *benchmark*). Logo, se selecionada uma aplicação que não pode ser executada na(s) plataforma(s) descritas nos passos anteriores, ela não será considerada para as próximas etapas. Uma vez selecionados esses requisitos básicos, as plataformas serão geradas (compiladas e armazenadas em formato executável). A subdivisão 5 (em verde) contém todas as plataformas geradas e a subdivisão 6 (em laranja) contém os arquivos produzidos pela simulação de cada uma delas. As ações possíveis após a configuração são: **Build** (para gerar todas as plataformas, **Execute** para simulá-las. Além disso, na subdivisão 7 (em amarelo), os itens do menu oferecem informações de ajuda para o uso da ferramenta. Já os botões **Delete** serve para apagar plataformas que foram construídas e que estão dispostos na subdivisão 5, e **Quit** para sair do simulador.

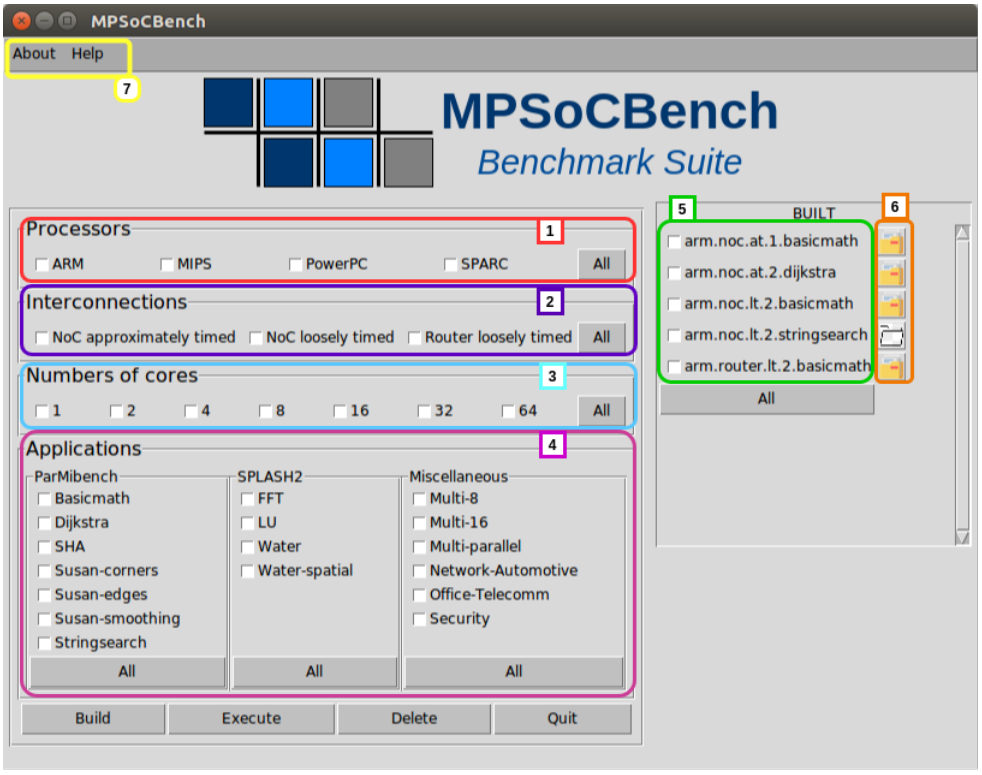

Figura 2. Interface em Desenvolvimento

Como exemplo, suponha a escolha dos 4 modelos de processadores, 3 tipos de interconexão, todas as possibilidades de configurações *multicore* (de 1 a 64 *cores*) e todas as possíveis aplicações. Essa simples escolha feita por 3 cliques nos botões All associada às 3 subdivisões, geraria 84 plataformas distintas que poderiam ser avaliadas utilizando grande parte das aplicações, totalizando mais de 1400 simuladores distintos. Além disso,

o controle de quais simulações já foram realizadas e quais ainda estão aguardando pode ser acompanhada pelo usuário por meio das subdivisões 5 e 6 da Figura 2.

Todos os dados estatísticos de saída do simulador são armazenados em formato de texto no arquivo **local_report.txt** no mesmo diretório em que o simulador foi criado. Porém, para simplificar a construção de gráficos e tabelas para análise dos dados, estabelecemos a geração de tabelas .csv contendo os principais dados de saída e facilitando a manipulação dos dados para elaboração de gráficos e tabelas.

4. Resultados Experimentais

Esta seção ilustra os resultados de alguns experimentos realizados com MPSoCBench já com as novas funcionalidades. Todos os experimentos foram feitos sobre a interconexão do tipo *crossbar* ou *router*, por ter simulação mais rápida.

A Tabela 1 mostra a média do tempo de simulação em segundos, entre três execuções de cada plataforma para as quatro ISAs e todas as aplicações em que a simulação é possível. Para facilitar a visualização, informações foram separadas em *cores* e de acordo com o ISA escolhido. As células vazias correspondem às plataformas inviáveis. Seguem alguns exemplos: para executar a aplicação **fft** pode ser executada com no máximo 16 *cores*, então as células referentes à plataforma de 32 ou 64 *cores* para essa aplicação estão vazias; outro exemplo, a aplicação **multi_8** contém 8 distintas aplicações que podem ser executadas em plataformas de 8 *cores* sem cooperação entre elas; e ainda, a aplicação **stringsearch** tem suporte para execução em plataformas de até 64 *cores*.

Com o objetivo de fazer análises mais detalhadas, reduzimos os experimentos para as seguintes combinações: arquitetura MIPS; com as sete possibilidades de quantidade de *cores* disponíveis (1 a 64); interconexões do tipo *crossbar* (router); com as aplicações de Basicmath, Dijkstra e SHA. Ao todo, resultando em 21 simuladores.

Foram extraídos os valores de quantidade de instruções executadas em cada aplicação e a estimativa de tempo execução. Este último valor difere do tempo de simulação exposto na Tabela 1. O tempo de simulação corresponde ao tempo em que o simulador demorou para executar. O tempo estimado de execução é a estimativa de tempo de execução da aplicação na plataforma descrita (ou seja, se a plataforma fosse real, qual o tempo que levaria para executar tal aplicação). Os dados estão dispostos nos gráficos da Figura 3.

Basicmath: É possível ver que na execução da aplicação Basicmath se mantém em um aumento gradativo de instruções, e quando está com 8 núcleos consegue atingir seu menor tempo.

Dijkstra: Com relação à aplicação Dijkstra, obteve seu o melhor tempo estimado de execução com 8 *cores*, sendo assim a configuração mais adequada quando se analisa a compensação entre o aumento de 206% da quantidade de instruções, com *simulation advance* que teve uma diminuição de 72% do tempo, sendo o tempo mais eficiente.

SHA: Ao verificar os valores apresentados, não é interpretado de imediato o favorecimento quando é acrescentado núcleos, mas ao analisar o funcionamento do algoritmo foi verificado que por se tratar de uma ferramenta usada na criptografia, é necessário o uso de arquivos externos. Com isso foi buscado nos dados de cada simulação a quantidade de

arquivos modificados em todas as 7 variações, visto que cada *core* precisa de um arquivo de entrada e foi feito uma normalização, que pode ser visto no Gráfico 2 da Figura 3.

PowerPC *MIPS* *SPARC* *ARM*

	Applications	01	02	04	08	16	32	64	01	02	04	08	16	32	64
Cooperative	Basicmath	2	3	5	11	21	56	139	2	3	5	10	21	54	121
		2	6	10	17	36	78	192	2	3	9	10	21	48	133
	LU	13	18	29	57	116	-	-	11	16	25	21	103	-	-
		23	29	44	87	174	-	-	26	33	49	94	188	-	-
	SHA	1	2	3	7	14	36	127	1	2	4	8	17	45	146
		1	2	3	7	16	42	143	1	2	4	9	20	52	174
	Water	25	38	54	73	303	-	-	32	33	49	67	275	-	-
		40	38	52	72	297	-	-	45	44	62	84	323	-	-
	Dijkstra	1	1	2	2	5	13	40	1	1	1	2	5	13	42
		1	1	1	2	6	15	46	1	1	1	2	5	12	45
	FFT	24	32	51	93	173	-	-	21	29	44	80	152	-	-
		32	43	66	119	215	-	-	39	54	86	143	261	-	-
	Stringsearch	19	19	20	21	23	28	45	29	29	30	30	31	35	44
		21	22	22	23	24	28	38	29	29	29	30	31	35	45
	Susan-corners	17	18	19	21	29	66	-	19	20	20	21	30	65	-
		28	28	29	30	39	72	-	26	26	27	30	39	72	-
	Susan-edges	33	42	48	53	59	114	-	35	44	50	55	58	109	-
		83	104	118	143	136	221	-	57	69	78	84	90	160	-
	Susan-smoothing	100	101	107	138	-	-	-	19	20	20	27	-	-	-
		33	33	38	52	-	-	-	20	20	22	30	-	-	-
	Water-Spatial	248	250	261	264	-	-	-	217	218	223	234	-	-	-
		351	353	369	380	-	-	-	209	198	211	233	-	-	-
Multi	Multi-parallel	-	-	22	42	113	641	2170	-	-	27	53	110	592	2046
		-	-	25	54	125	520	1707	-	-	30	57	115	227	491
	Office-Telecomm	-	-	68	-	-	-	-	-	-	202	-	-	-	-
		-	-	199	-	-	-	-	-	-	100	-	-	-	-
	Network-Automotive	-	-	115	-	-	-	-	-	-	69	-	-	-	-
		-	-	87	-	-	-	-	-	-	89	-	-	-	-
	Security	-	-	381	-	-	-	-	-	-	355	-	-	-	-
		-	-	348	-	-	-	-	-	-	412	-	-	-	-
	Multi-8	-	-	-	542	-	-	-	-	-	-	512	-	-	-
		-	-	-	438	-	-	-	-	-	-	555	-	-	-
	Multi-16	-	-	-	-	940	-	-	-	-	-	-	729	-	-
		-	-	-	-	560	-	-	-	-	-	-	562	-	-

Tabela 1. Média do tempo de simulação, em segundos, entre três execuções de cada plataforma, usando a interconexão router.lt, para as quatro ISA's disponíveis.

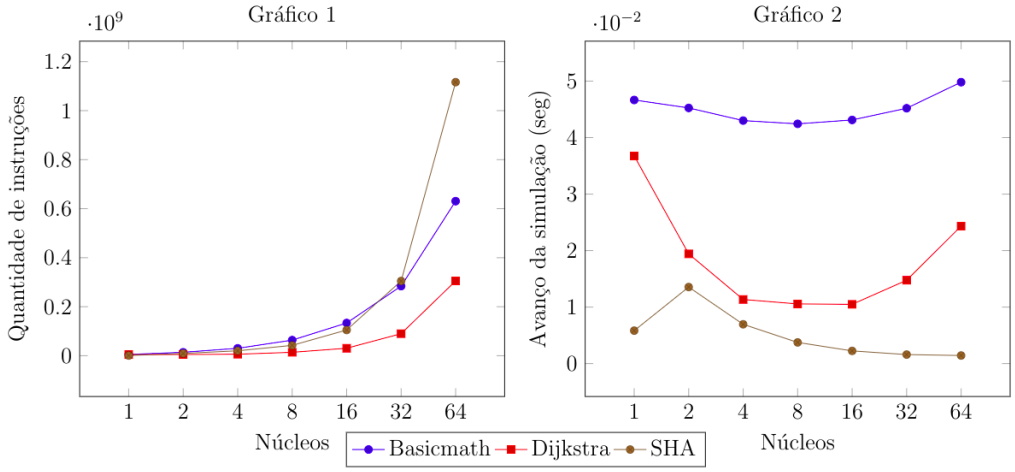

Figura 3. Análise de quantidade de instruções por *cores*

4.1. Simuladores funcionais correlatos

Existem muitos outros simuladores de sistemas computacionais utilizados em pesquisas científicas [Phillips et al. 1979, Binkert et al. 2011] e também para o ensino [Vollmar and Sanderson 2006, Patel et al. 2011]. Quando comparado a outros simuladores mais robustos voltados para pesquisa, há uma vantagem significativa do MPSoCBench considerando a sua nova interface com usuário. Para simuladores com finalidades de apoio ao ensino, os quais comumente apresentam uma interface mais amigável, a vantagem do MPSoCBench é na diversidade de configurações arquiteturais permitida pela ferramenta. Entretanto, não foi possível realizar um estudo comparativo mais detalhado entre as ferramentas no contexto de IHC (interação humano-computador).

5. Considerações Finais

Este artigo apresentou a metodologia e os resultados da implementação de novas funcionalidades ao *framework* MPSoCBench. As novas funcionalidades tem foco em tornar mais amigável e simples o uso da ferramenta por usuários de cursos de Arquitetura de Computadores e afins, em nível de graduação ou pós-graduação. O artigo também demostrou como a ferramenta pode ser usada e como dados extraídos da simulação podem auxiliar na validação de conceitos e tópicos desta área da computação.

Os experimentos mostrados representam apenas uma ínfima parte do que é possível realizar por meio da ferramenta. Além disso, também é compromisso futuro facilitar a configuração de parâmetros de cache (como políticas de cache, políticas de substituição de blocos e tamanhos dos blocos), configuração de tamanho de memória, entre outros aspectos arquiteturais e microarquiteturais ainda não explorados no trabalho atual. Além disso, pretende-se dar suporte para execução dos simuladores gerados em um ambiente HPC (do inglês, *high performance computing)*.

Até o momento de envio deste artigo a Interface estava passando por um processo de testes das suas funcionalidades por ter sido desenvolvida no decorrer da Iniciação Científica, então ainda não há estatísticas de avaliações externas. Entretanto, há uma pretenção em aplicá-la no transcorrer da disciplina de Arquitetura de Computadores em nível de graduação em 2020/2 e em nível de pós-graduação em 2021/1,

gerando dados para estudos e análises futuras. Pretende-se avaliar se a nova interface permitirá sanar as dificuldades encontradas pelos usuários dos trabalhos realizados em [Duenha and Azevedo 2016] e [Duenha et al. 2017].

Referências

(2012). IEEE Std 1666^{TM} Standard SystemC Language Reference Manual. IEEE Computer Society.

Azevedo, R., Rigo, S., Bartholomeu, M., Araujo, G., Araujo, C., and Barros, E. (2005). The ArchC Architecture Description Language and Tools. In *International Journal of Parallel Programming. Vol. 33, No. 5*, pages 453–484.

Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S., et al. (2011). The gem5 simulator. *ACM SIGARCH computer architecture news*, 39(2):1–7.

Duenha, L. and Azevedo, R. (2016). Utilização dos simuladores do mpsocbench para o ensino e aprendizagem de arquitetura de computadores. In IEEE, editor, *International Journal of Computer Architecture Education (IJCAE), 2016*, pages 26–31.

Duenha, L., Crominski, F., Santos, M. T., and Ribeiro, R. (2017). Avaliação de preditores de desvios por meio de simuladores como parte do processo de ensino e aprendizagem de arquitetura de computadores. *International Journal of Computer Architecture Education (IJCAE)*, 6(1):1–9.

Duenha, L., Guedes, M., Almeida, H., Boy, M., and Azevedo, R. (2014). Mpsocbench: A toolset for mpsoc system level evaluation. In *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV), 2014 International Conference on*, pages 164–171. IEEE.

Duenha, L., Madalozzo, G., Santiago, T., Moraes, F., and Azevedo, R. (2016). Mpsocbench: A benchmark for high-level evaluation of multiprocessor system-on-chip tools and methodologies. *Journal of parallel and distributed computing*, 95:138–157.

Patel, A., Afram, F., Chen, S., and Ghose, K. (2011). Marss: A full system simulator for multicore x86 cpus. In *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1050–1055.

Phillips, D. T., Handwerker, M., and Hogg, G. L. (1979). Gems: A generalized manufacturing simulator. *Computers & Industrial Engineering*, 3(3):225–233.

Santos, R., Duenha, L., Silva, A. C., Sousa, M., Tedesco, L. A., Melgarejo, J. C., Santos, T., Azevedo, R., and Moreno, E. (2018). Dark-silicon aware design space exploration. *Journal of Parallel and Distributed Computing*, 120:295–306.

Tkinterbook (2005). An introduction to tkinter. `http://effbot.org/tkinterbook/`. Accessed on 2019-20-11.

Vollmar, K. and Sanderson, P. (2006). Mars: an education-oriented mips assembly language simulator. In *Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 239–243.