COMPARAÇÃO DO ERRO COMPUTACIONAL EM SIMULAÇÃO DE UM SISTEMA CAÓTICO EM DISPOSITIVOS ARM E INTEL

WENDERSON DE SOUZA SILVA[*], EDUARDO PINTO MAGALHÃES[*], ERIVELTON G. NEPOMUCENO[*]

[*] *Programa de Pós-Graduação em Engenharia Elétrica (UFSJ/CEFET)*
UFSJ - Universidade Federal de São João del-Rei
Pça. Frei Orlando, 170 - Centro - 36307-352 - São João del-Rei, Minas Gerais, Brasil

Emails: wenssilva@hotmail.com, eduardopintomagalhaes@gmail.com, nepomuceno@ufsj.edu.br

Abstract— This work deals with the differences found in simulations of chaotic systems (Lorenz equations) using ARM processors compared to the results obtained using Intel. In order to make a comparison of the results, was used the Lower Bound Error and calculated the loss of significant digit of the simulation given the variation of the step, and it was observed that the decrease of the step to the cost of longer processing time does not always result in the better results.

Keywords— Lorenz, ARM, Intel, Chaotic Systems, Lower Bound Error, Floating Point

Resumo— Este trabalho trata das diferenças encontradas nas simulações de sistemas caóticos (Equações de Lorenz) utilizando processa-dores ARM em comparação com o resultado obtido utilizando Intel. Para efetuar a comparação dos resultados foi utilizado o Erro de Limite Inferior e calculado a perda de algarismos significativos da simulação dada a variação do passo, sendo observado que a diminuição do passo ao custo de maior tempo de processamento nem sempre resulta em melhores resultados.

Palavras-chave— Lorenz, ARM, Intel, Sistemas caóticos, Erro de Limite Inferior, Ponto Flutuante

1 Introdução

A simulação numérica em processadores x86 vem sendo estudada ao longo dos últimos anos, com ênfase para os erros gerados pelo cálculo com ponto flutuante (Nepomuceno and Mendes, 2017), a repetibilidade e confiabilidade dos resultados obtidos utilizando diferentes CPUs (Šalamon and Dogša, 2007), ou até mesmo em GPUs (Taufer et al., 2010).

Apesar dos *chips* Intel terem conquistado o mercado de computadores pessoais e servidores, em detrimento aos AMD e IBM, atualmente tem-se observado um aumento dos dispositivos que utilizam processadores ARM principalmente em *tablets*, *smartphones* e dispositivos IoT, mas também em equipamentos como servidores e *notebooks*, antes dominados pela Intel (Forbes, 2017).

Na Amazon americana, *Chromebooks* com chip ARM já aparecem na lista de notebooks mais vendidos (Amazon, 2017). A participação de servidores ARMs apesar de pequena, da ordem de 1%, possui tendência de crescimento (Forbes, 2017). Tais fatos levaram a Microsoft a desenvolver em parceria com a Qualcomm uma versão do Windows especifica para processadores ARM demostrada na *build* 2017 e lançada no segundo semestre de 2017.

Dado a importância e perspectiva futura da arquitetura ARM, nos últimos anos surgiram alguns artigos utilizando estes processadores em simulações numéricas, tanto de Equações Diferenciais (Sala, 2012) quanto para Inteligência Computacional (Gokhale et al., 2014; Sakr et al., 2017; Lane et al., 2016). Este trabalho visa analisar o impacto da mudança da arquitetura do processador de Intel para ARM no erro do resultado obtido na simulação, semelhante ao que foi feito em (Šalamon and Dogša, 2007) para Intel x86. Para isto foram utilizados algoritmos em Octave de simulações numéricas de sistemas não-lineares com comportamento caótico em *smartphones* com sistema operacional Android, de diferentes fabricantes, visando fazer uma análise da variação do resultado quanto a arquitetura do processador utilizado ARM ou Intel x86 e x86-64. O sistema caótico utilizado são as equações de Lorenz (Sparrow, 1982), desenvolvida em 1963 por Eduard Norton Lorenz. Suas equações levaram ao primeiro atrator caótico e serviram como base para muitos trabalhos na área de sistemas não lineares tal como pode ser visto em (Lü and Chen, 2002). A escolha por um sistema caótico é pela natureza do mesmo, neste tipo de sistema um pequena diferença nas condições iniciais podem levar a resultados completamente diferentes potencializando qualquer divergência nos cálculos com ponto flutuante realizados na arquitetura Intel ou ARM.

Essa linha de pesquisa tem-se tornado cada vez mais explorada, o que dá relevância à novas descobertas sobre comportamentos de sistemas caóticos. Como exemplo, na literatura tem-se encontrados trabalhos que deram importância a essa linha de raciocínio, tais como: o controle nebuloso que usa o conceito de compensação paralela (Costa et al., 2004); a influência de *softwares* e sistemas operacionais na simulação de modelos não-lineares (Milani et al., 2016); a implementação de um mapa sob o padrão de 32 bits tendo como objeto de estudo um sistema caótico (Silva et al., 2017) e a simulação de sistemas dinâmicos com análise intervalar que traz uma abordagem de incertezas numéricas (Peixoto et al., 2016).

Na análise dos resultados obtidos será utilizado o "Erro de limite inferior" proposto em Nepomuceno e Martins (2016), deste modo serão utilizadas duas variações do Runge-Kutta de 4ª ordem de maneira que se encontre duas pseudo-orbitas. Calculando o logaritmo da diferença entre as pseudo-orbitas será obtido um

dado qualitativo para comparação do resultado de cada simulação. As análises consistem em variar o passo de integração, observar o erro da simulação quanto a perda de algarismos significativos tanto no mesmo processador usando a função original e uma extensão, quanto a simulação de uma mesma equação apenas modificando o dispositivo.

O trabalho está organizado da seguinte forma. Na seção 2 são apresentados conceitos sobre os Sistemas Caóticos, a Equação de Lorenz, o Erro de Limite Inferior, o Tempo Crítico, e o *software* GNU Octave/Octave. Na seção 3, a metodologia proposta descreve os detalhes de como foi realizada a implementação do algoritmo nas arquiteturas ARM e Intel, bem como a aplicação das Equações de Runge-Kutta e a utilização do Erro de Limite Inferior. Na seção 4, os resultados das arquiteturas ARM, Intel x86 e Intel x84-64 são apresentados e comparados e na seção 5 são apresentadas as conclusões.

2 Conceitos Preliminares

2.1 Sistemas Caóticos

Usualmente, é utilizado aproximação e linearização na resolução de problemas com sistemas dinâmicos, isto simplifica o equacionamento e determina a solução dos problemas de maneira mais rápida. A natureza por outro lado possui essencialmente características não lineares, fazendo com que o processo de linearização adicione erros a solução.

Em 1955, Eduard Norton Lorenz não satisfeito com os resultados ao realizar simulações para previsão do tempo desenvolveu um modelo não-linear para o sistema. Após vários testes computacionais Lorentz percebeu que em alguns momentos o resultado divergia devido ao erro de arredondamento, mesmo utilizando as mesmas condições iniciais. Sistemas como o encontrado por Lorentz onde pequenas variações modificam totalmente a saída do sistema a longo prazo são denominados sistemas caóticos (Andrucioli, 2008).

2.2 Equação de Lorenz

Em 1963, após simplificações das equações de convecção de Navier-Stokes utilizadas em mecânica dos fluidos (Sparrow, 1982), o sistema de Lorentz tornou-se composto por três equações diferencias não-lineares da seguinte maneira, onde x, y e z são os eixos do sistema de coordenadas cartesiano, ρ, σ e β são as constantes do sistema e t é a unidade de tempo de Lorenz (LTU):

$$\frac{\partial x}{\partial t} = \sigma (y - x) \tag{1}$$

$$\frac{\partial y}{\partial t} = x(\rho - z) - y \tag{2}$$

$$\frac{\partial z}{\partial t} = x \cdot y - \beta \cdot z \tag{3}$$

O sistema se torna caótico quando $\rho = 24.74$, $\sigma = 10$ e $\beta = 8/3$. (Jambersi) (Andrucioli, 2008)

2.3 Runge–Kutta de 4ª Ordem

O Runge-Kutta de quarta ordem é um método iterativo para resolução de equações diferenciais. O procedimento consiste em encontrar as constantes $K1$, $K2$, $K3$ e $K4$, dada as seguintes equações, onde h é o passo de integração.

$$y_{n+1} = y_n + \frac{1}{6}(k1 + 2 \cdot k2 + 2 \cdot k3 + k4) \tag{4}$$

$$k1 = h \cdot f(x_n, y_n) \tag{5}$$

$$k2 = h \cdot f\left(x_n + \frac{1}{2} \cdot h, y_n + \frac{1}{2} \cdot k1\right) \tag{6}$$

$$k3 = h \cdot f\left(x_n + \frac{1}{2} \cdot h, y_n + \frac{1}{2} \cdot k2\right) \tag{7}$$

$$k4 = h \cdot f(x_n + h, y_n + k3) \tag{8}$$

2.4 Erro de Limite Inferior e Tempo Crítico

Os conceitos a seguir seguem o estudo dado em (Nepomuceno e Mendes, 2017) aplicando o erro de limite inferior a sistemas não-lineares de tempo continuo.

Definição 1: Orbita é uma sequência de valores em um mapa logístico, da maneira:

$$\{x_n\} = [x_1, x_2, x_3, ..., x_n] \tag{9}$$

Definição 2: A função f é uma extensão da função g, se estas forem equivalentes, mas através do uso de propriedades matemáticas as funções f e g se apresentam com estruturas diferentes. Por exemplo fazendo uso de propriedade distributiva a Equação 10 é matematicamente equivalente a Equação 4, mas apresentada em outra forma, tornando assim (10) uma extensão de (4).

$$y_{n+1} = y_n + \frac{1}{6}(k1 + 2 \cdot (k2 + k3) + k4) \tag{10}$$

Definição 3: O Erro de limite inferior (δ) é dado como a metade do módulo da diferença entre duas pseudo-orbitas resultantes de duas extensões de uma dada função, ou seja:

$$\delta_{\alpha,n} = \frac{\left|\widehat{a}_{\alpha,n} - \widehat{a}_{\beta,n}\right|}{2} \tag{11}$$

2.5 *GNU Octave/Octave*

A GNU Octave é uma linguagem computacional de alto nível destinado a computação científica. Ela é projetada para resolver problemas numéricos lineares e não lineares, matrizes, vetores e para criar sofisticados gráficos. Para o desenvolvimento de um programa baseado nessa linguagem utiliza-se o *software* Octave. O Octave é desenvolvido em C++, utiliza bibliotecas STL, um interpretador para executar os *scripts* e é extensível a partir do carregamento de módulos. A escolha dessa ferramenta baseou-se em fatores como: quantidade de funções necessárias para realizar problemas numéricos não lineares e por principalmente ser baseada na filosofia de *software* livre (Selhofer et al., 2008).

3 Metodologia

Neste trabalho a implementação do código é feita em Octave utilizando números em ponto flutuante com precisão *double*. A compilação do algoritmo nos *smartphones* Android é realizada por meio do aplicativo GNURoot Octave disponível na Play Store, esse *software* é *opensource* que pode ser encontrado no site *github*. Ele utiliza um emulador de terminal Linux, acrescido do Octave em sua versão 3.8.2 otimiza-dos para processadores ARM.

Nos testes com computadores Intel 32 bits é utilizado o sistema operacional Lubuntu, mantendo o Octave na mesma versão utilizada anteriormente. Nos testes com o Intel 64 bits a versão utilizada é a 4.2.1. Os demais procedimentos são iguais para ARM e Intel.

As equações de Lorenz são simuladas através do Runge-Kutta de 4ª ordem e sua extensão, como apresentado nas equações 4 e 9, com passo de integração da variando entre 10^{-2} e 10^{-4} com 100 segundos de tempo de simulação. Deste modo são geradas duas pseudo-orbitas do sistema com os parâmetros $\rho = 28$, $\sigma = 10$ e $\beta = 8/3$. Os dados obtidos são salvos em um arquivo .mat logo após a simulação em cada dispositivo e guardados em um banco de dados.

Posteriormente é utilizado o Erro de limite inferior apresentado na Seção 2.4 para comparar os resultados e analisar a perda de algarismos significativos em cada simulação. Deste modo é gerado o gráfico logaritmo da diferença entres as pseudo-orbitas pelo tempo em segundos e obtido o tempo máximo de simulação e a curva do erro para cada arquitetura. (Nepomuceno e Mendes, 2017).

De posse dos dados são analisadas as diferenças nos resultados quanto a variação no processador ARM ou Intel, a arquitetura 32/64 bits e a influência da variação do passo de integração do Runge-Kutta de 4ª ordem.

4 Resultados

Após a coleta dos dados, o primeiro passo foi confirmar as diferenças esperadas em cada simulação. Na Figura 1 é possível observar que apesar de utilizar o mesmo *software* a variação da arquitetura do processador utilizada faz com que após algumas iterações o resultado tome valores completamente diferentes, não sendo possível confiar nos resultados obtidos.

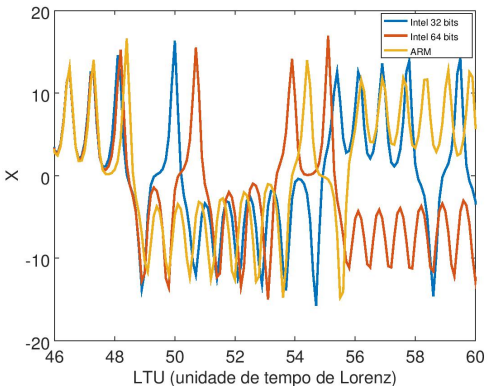

Figura 1: Gráfico da resposta de X, dado pela Equação 1, para a simulação no Octave em sistemas Intel 32/64 bits e ARM com passo de integração de 10^{-4}.

4.1 *Intel 32 bits vs 64 bits*

Foi observado que as simulações utilizando a arquitetura x86 possui resultados ligeiramente melhores que a mesma simulação utilizando a arquitetura x86-64. Na Figura 2 está o gráfico de Erro de Limite Inferior e é possível notar que a perda de algarismos significativos ocorre mais lentamente para o Intel 32 bits.

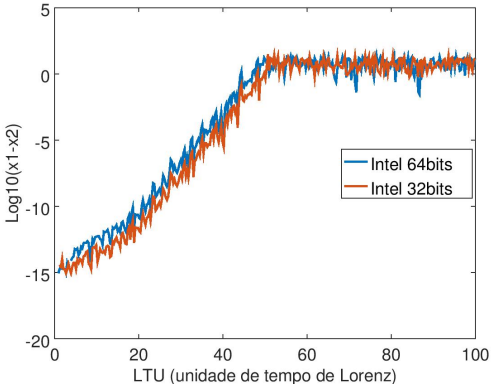

Figura 2: Gráfico do Erro de Limite Inferior, gerado a partir da resposta de X, Equação 1 e sua extensão, simulação usando o Octave 4.2.1 na arquitetura Intel 32 bits e Intel 64 bits com passo de integração de 10^{-4}.

4.2 *Intel 32 bits vs ARM*

Na comparação dos resultados obtidos com arquitetura x86 e ARM não foi observado variações relevan-

tes no resultado quando considerado apenas o tempo máximo de simulação proposto pelo Limite de Erro Inferior, Figura 3. No entanto, por se tratar de um sistema caótico, com o aumento do número de iterações as diferenças entre os resultados também crescem.

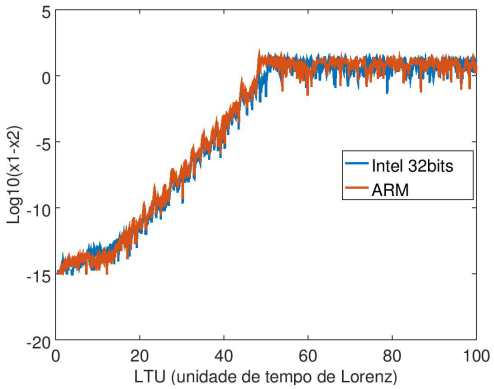

Figura 3: Gráfico suavizado do Erro de Limite Inferior, gerado a partir da resposta de x, Equação 1 e sua extensão, simulação usando o Octave 3.8.2 na arquitetura Intel 32 bits e ARM com passo de integração de 10^{-4}.

4.3 Variação do Passo de Integração

Com a variação do passo de integração de 10^{-2} a 10^{-4} nas simulações feitas na arquitetura ARM pode-se observar que quanto menor o passo de integração menor é a perda de algarismos significativos e consequentemente maior o tempo de confiabilidade da simulação, o resultado pode ser observado na Figura 4. Contudo, essa afirmação não é sempre verdadeira, havendo situações onde diminuir o passo leva a uma piora nos resultados.

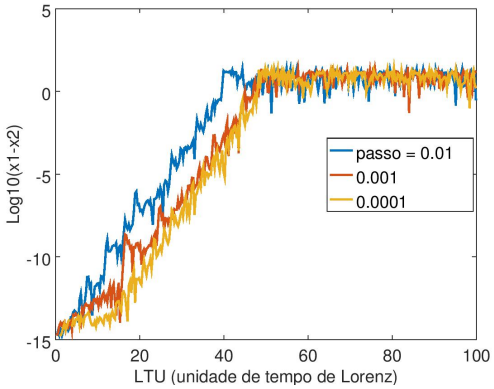

Figura 4: Gráfico suavizado do Erro de Limite Inferior, gerado a partir da resposta de X, Equação 1 e sua extensão, simulação usando o Octave 3.8.2 na arquitetura ARM.

Foi ainda observado que, quando se compara com os resultados obtidos em outras arquiteturas a perda de algarismos significativos ocorre da mesma maneira, tanto no Intel 32 quanto no ARM. É importante observar que o ganho no tempo de simulação diminuindo o tamanho do passo não é linear se mantendo praticamente constante de 10^{-3} para 10^{-4} , como pode ser observado na Tabela 2. Cada linha indica uma casa decimal, iniciando do 1 que é o algarismo de unidades, 2 é o primeiro algarismo após a virgula, as colunas indicam o passo utilizado na simulação.

Tabela 1: Tempo (dias) de simulação até o erro no primeiro algarismo após a virgula.

Passo de Integração	Intel 32	ARM
10^{-2}	32,242	37,167
10^{-3}	43,254	43,278
10^{-4}	44,15	44,06

Com a intenção de observar o comportamento da perda de algarismos com a variação do passo de integração, foi realizado várias simulações comparando a perda de algarismos entre os resultados utilizando processador ARM e Intel x64 variando o passo de 10^{-2} a 10^{-4} e observando a perda dos 3 primeiros algarismos após a virgula para cada uma das simulações gerado um gráfico conforme ilustrado na Figura 5. É possível observar que para um passo de integração menor que 0,001 o ganho de tempo de simulação permanece dentro de uma faixa de valores, não havendo mais ganhos significativos.

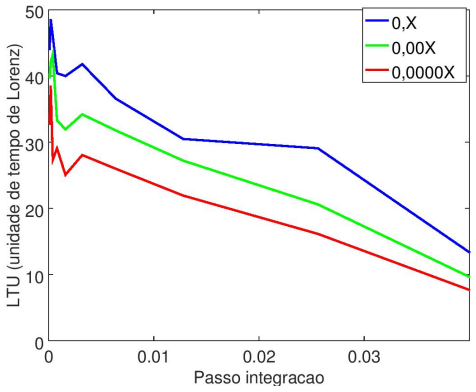

Figura 5: Gráfico relacionando o tempo de simulação usando ARM vs Intel 64 até a perda do dado algarismo em relação ao passo de integração utilizado.

Tendo como base que o tempo de simulação obteve valores muito próximos quando utilizado os passos 10^{-3} e 10^{-4} foi realizado um novo teste variando o passo em dez estágios dentro deste intervalo, utilizando o Erro de Limite Inferior para comparar os valores utilizando chips ARM e Intel x86-64.

Observando o resultado na Tabela 1, é possível constatar que não obrigatoriamente a redução no passo de integração reflete em um melhor resultado. Um ponto relevante desta análise é uma simulação com 4 casas decimais após a virgula, linha 5 da Tabela 2,

Tabela 2: Tempo (dias) de simulação até o erro no algarismo indicado considerando o Erro de Limite Inferior para a Equação 1 comparando a simulação em arquitetura ARM vs Intel x86-64.

	10^{-4}	2.10^{-4}	3.10^{-4}	4.10^{-4}	5.10^{-4}	6.10^{-4}	7.10^{-4}	8.10^{-4}	9.10^{-4}	10^{-3}
1	47,110	52,316	44,501	51,903	48,302	44,909	47,119	42,874	43,122	47,027
2	43,930	48,543	44,235	45,986	43,736	42,619	43,234	40,394	39,087	44,560
3	41,659	46,537	42,478	45,726	43,463	39,937	42,953	37,930	36,621	40,612
4	39,636	41,985	39,243	43,226	40,232	37,459	39,775	33,290	34,840	37,444
5	34,963	41,701	36,694	40,016	37,156	33,575	36,728	30,988	31,689	35,903
6	32,637	38,503	32,137	27,329	33,097	29,026	34,064	29,049	29,935	33,926
7	30,858	33,807	30,615	27,102	32,815	28,725	32,326	26,591	26,748	31,443
8	27,490	33,491	27,853	26,862	28,935	25,539	29,256	24,103	24,218	28,912
9	25,729	30,266	20,556	22,442	25,765	23,058	27,333	22,328	20,448	24,401
10	23,228	27,775	20,309	22,007	24,787	20,610	24,092	19,316	18,715	16,442
11	20,763	25,218	17,321	19,500	15,937	17,386	22,325	17,299	16,780	16,203
12	17,668	17,946	15,340	16,352	15,690	14,877	18,564	13,395	13,500	15,943
13	14,700	17,698	11,231	13,836	11,965	11,983	14,372	9,628	11,089	13,496
14	7,829	4,826	6,225	4,262	4,168	4,863	7,576	3,079	7,137	6,254

onde a simulação com passo 10^{-4} possui desempenho pior que a simulação com 10^{-3}. Neste casso a redução no passo de integração levou a dez vezes mais iterações, consumindo recurso computacional e sobretudo tempo e o resultado encontrado é pior do que o anterior.

5 Conclusão

Neste trabalho foi realizado uma comparação do erro computacional para simulação das equações de Lorenz em diferentes arquiteturas de sistemas. O resultado da análise mostra claramente uma diferença nos resultados obtidos com a mudança do processador de Intel para ARM, sendo que os resultados para ARM e Intel 32 bits são mais próximos.

É observável pela Figura 5 que o passo de integração escolhido influencia diretamente na repetibilidade do resultado quando comparado a simulação realizada em Intel x86-64 ou em ARM. Por meio da Tabela 1 nota-se que dado a característica caótica do sistema simulado, a diminuição do passo de integração nem sempre gera resultados melhores, sendo nítido o problema quando se observa a linha 6 da Tabela para os passos 10^{-3} e 10^{-4} onde apesar de utilizar um passo dez vezes menor e consequentemente dez vezes a mais de interações e tempo de processamento o resultado obtido com o passo menor é ligeiramente melhor.

É ainda importante ressaltar que durante os testes, não foi observado nenhuma diferença entre as simulações usando processadores ARM de diferentes fabricantes, diferente do observado por Salamon (2007) para processadores baseados em x86. Sendo ainda necessário um estudo mais aprofundado quanto ao tema e em relação também a arquitetura ARM64 não sendo conclusivo neste trabalho se há ou não diferenças nestas simulações.

Em trabalhos futuros pretende-se analisar além do ARM64 a influência da mudança de arquitetura em outros *softwares* de simulação numérica como Python e outros sistemas caóticos, por exemplo os sistemas de Chua e as equações de Chen.

Agradecimentos

Os autores agradecem, pelo suporte financeiro e ao Programa de Pós-Graduação em Engenharia Elétrica, associação ampla UFSJ/CEFET-MG.

Referências

Amazon (2017). Best sellers eletronics laptop computers in 10/09/17, `www.amazon.com/Best-Sellers-ElectronicsLaptop-computers/zgbs/electronics/565108#2`.

Costa, W. T., Nepomuceno, E. G. and NETO, O. M. (2004). Controle nebuloso de sistemas não-lineares via método takagi-sugeno: Uma abordagem didática usando simulink, *XV Congresso Brasileiro de Automática*.

Forbes (2017). Server cpu predictions for 2017, `www.forbes.com/sites/moorinsights/2017/01/10/server-cpu-predictions-for-2017/#b9f3d1165a75`.

Gokhale, V., Jin, J., Dundar, A., Martini, B. and Culurciello, E. (2014). A 240 g-ops/s mobile coprocessor for deep neural networks, *Proceedings*

of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 682–687.

Lane, N. D., Bhattacharya, S., Georgiev, P., Forlivesi, C., Jiao, L., Qendro, L. and Kawsar, F. (2016). Deepx: A software accelerator for low-power deep learning inference on mobile devices, *Information Processing in Sensor Networks (IPSN), 2016 15th ACM/IEEE International Conference on*, IEEE, pp. 1–12.

Lü, J. and Chen, G. (2002). A new chaotic attractor coined, *International Journal of Bifurcation and chaos* **12**(03): 659–661.

Milani, F. L., Junior, W. R. L., Martins, S. A. M. and Nepomuceno, E. G. (2016). Influência de softwares e sistemas operacionais na simulação de modelos dinâmicos não lineares, *XXI Congresso Brasileiro de Automática*.

Nepomuceno, E. G. and Mendes, E. M. (2017). On the analysis of pseudo-orbits of continuous chaotic nonlinear systems simulated using discretization schemes in a digital computer, *Chaos, Solitons & Fractals* **95**: 21–32.

Peixoto, M., Nepomuceno, E., Junior Rodrigues, H., Martins, S. and Amaral, G. (2016). Simulação de sistemas dinâmicos com análise intervalar: Um estudo de caso com o circuito rlc, *XXI Congresso Brasileiro de Automática*.

Sakr, C., Kim, Y. and Shanbhag, N. (2017). Analytical guarantees on numerical precision of deep neural networks, *International Conference on Machine Learning*, pp. 3007–3016.

Sala, F. A. (2012). Mobile phone as a platform for numerical simulation, *Computer Physics Communications* **183**(1): 26–29.

Šalamon, M. and Dogša, T. (2007). Neponovljivo obnašanje programov na različnih procesorjih.

Selhofer, H., Oliver, M. and Scofield, T. (2008). Introduction to gnu octave.

Silva, D. A., Pereira, E. B. and Nepomuceno, E. G. (2017). Realização do mapa logístico em fpga usando um padrão ponto fixo de 32 bits, *XIII Simpósio Brasileiro de Automação Inteligente*.

Sparrow, C. (1982). The lorenz equations: bifurcations, chaos, and strange attractors appl, *Math. Sci* **41**.

Taufer, M., Padron, O., Saponaro, P. and Patel, S. (2010). Improving numerical reproducibility and stability in large-scale numerical simulations on gpus, *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, IEEE, pp. 1–9.