Análise Comparativa de Síntese de Alto Nível para Algoritmos de Multiplicação de Matrizes em FPGA

Henrique Gregory Gimenez[1], **Edson Toshimi Midorikawa**[1]

[1]Laboratório de Arquitetura e Computação de Alto Desempenho
Departamento de Engenharia de Computação e Sistemas Digitais
da Escola Politécnica – Universidade de São Paulo (USP)

{henriquegregory,emidorik}@usp.br

Resumo. *Este artigo apresenta uma análise comparativa de algoritmos de multiplicação de matrizes (MM), denominados padrão (baseline) e em blocos (blocked), utilizando síntese de alto nível (HLS). Foram avaliados os tempos de execução com a placa FPGA PYNQ-Z2. Também foi estudada a alocação de recursos da FPGA em ambos os algoritmos após a síntese. Os resultados mostram que o algoritmo blocked em FPGA apresenta desempenho superior às demais versões para matrizes grandes, ao passo que também consome mais recursos conforme o tamanho das matrizes de entrada aumenta.*

1. Introdução

O processo de desenvolvimento de hardware tem crescido em complexidade, em eficiência e em demanda ao longo dos anos. A síntese de alto nível, *High Level Synthesis* (HLS), permite aos desenvolvedores de hardware aumentarem o nível de abstração e focarem suas atenções em questões arquiteturais dos projetos [Kastner et al. 2018]. O projetista escreve funções em linguagem de alto nível como C/C++ que sejam sintetizáveis em linguagem de descrição de hardware como Verilog/VHDL.

Este estudo busca realizar uma comparação entre dois algoritmos de MM: *baseline*, o algoritmo iterativo mais simples, e *blocked* ou *em blocos*, baseado na divisão das matrizes originais de entrada em submatrizes e operando o algoritmo *baseline* nelas. Foram levantadas métricas sobre o tempo de execução dos diferentes algoritmos, nas suas versões em hardware e em software, e sobre o uso dos recursos da FPGA ao sintetizá-los. As análises consideraram a influência do tamanho das matrizes de entrada sobre os resultados. Ademais, as implementações dos algoritmos foram retiradas do repositório de [Lu and Chen 2024], desenvolvido pela Xilinx.

A seção 2 discorre sobre trabalhos relacionados à multiplicação de matrizes em FPGA utilizando HLS. Em seguida, a seção 3 detalha o desenvolvimento e os resultados obtidos. Por fim, conclui-se o artigo na seção 4 e perspectivas para futuros estudos também são apresentadas.

2. Trabalhos Relacionados

Esta seção apresenta trabalhos relacionados a esta pesquisa, sendo a lista limitada aos três artigos a seguir devido à restrição de espaço.

Em [Skalicky et al. 2013], uma análise de custo-benefício foi realizada utilizando ferramentas de HLS em FPGAs para implementar três algoritmos de multiplicação de matrizes: o algoritmo padrão, o de Strassen e o de matrizes esparsas. É feita uma comparação

entre os hardwares gerados pelo Vivado com circuitos customizados. A FPGA utilizada no estudo foi a *Xilinx Virtex 6-475T*. Também foram realizados testes dos algoritmos em software com o processador *Intel Core i7 Sandy Bridge 3.4GHz*.

[Leon-Vega and Castro-Godinez 2023] propuseram uma arquitetura aceleradora genérica e configurável para *Generic Matrix Multiplication-Additions* (GEMMA), que consiste em obter uma matriz de saída a partir de uma multiplicação entre duas matrizes de entrada seguida da adição de uma terceira matriz. A implementação foi feita em C++ e as ferramentas *Vitis* e *Vivado* foram usadas para a síntese de alto nível. O algoritmo proposto visa diminuir o uso de recursos ao sintetizar em FPGA. A placa FPGA utilizada foi a *Avnet Zedboard (Xilinx ZYNQ XC7Z020)*.

O trabalho de [Meng et al. 2022] desenvolveu uma implementação de operações de matrizes de números complexos em ponto flutuante. A operação executada consiste em multiplicar uma matriz de coeficientes de tamanho NxN por um vetor de entrada Nx1 para gerar uma saída Nx1. Um algoritmo em C++ gera números aleatórios, realiza a operação e produz valores de referência e que posteriormente são utilizados para comparar os resultados obtidos no módulo em hardware.

3. Desenvolvimento e Resultados Experimentais

3.1. Algoritmos Utilizados

O algoritmo *baseline* consiste no algoritmo mais elementar de multiplicação de matrizes. Três *loops* aninhados iteram sobre cada elemento das linhas da primeira matriz de entrada e das colunas da segunda matriz de entrada, realizando o produto entre esses elementos que, posteriormente, são somados para obter uma única posição da matriz resultante.

Já a multiplicação de matrizes em blocos parte do mesmo princípio do algoritmo *baseline*, porém, para obter o produto final, as matrizes de entradas são particionadas. Por exemplo, para multiplicar duas matrizes de tamanhos 128x128, pode-se dividir a matriz A em quatro submatrizes de 128x32 e a matriz B em quatro submatrizes de 32x128 e realizar operações paralelamente [Lu and Chen 2024].

3.2. Metodologia

As ferramentas Xilinx *Vitis 2024.2* e *Vivado 2024.2* foram utilizadas para a geração do HDL a partir dos códigos bases em C++ e para a geração dos *bitstreams* programáveis na FPGA, respectivamente. Os resultados deste artigo foram obtidos utilizando a placa *PYNQ-Z2* que é equipada com uma FPGA *Zynq-7000 SoC XC7Z020-1CLG400C*, além de um processador *dual-core ARM Cortex-A9* que permitiu avaliar os algoritmos nas suas versões em software.

Aplicou-se a seguinte metodologia: a biblioteca *NumPy* gera 5 matrizes aleatórias para cada uma das entradas A e B. Para cada plataforma (SW ou HW) e algoritmo (*baseline* ou *blocked*), realiza-se a multiplicação de matrizes e verifica-se o tempo entre o início e o fim da operação. Ao fim do processo, obtém-se 5 medidas de tempo diferentes e a média aritmética é calculada. A biblioteca *NumPy* foi usada para gerar valores de referência. Assim, uma matriz de saída obtida pela biblioteca é selecionada para comparar com os valores de saída das demais plataformas, de modo a garantir a corretude das operações.

3.3. Avaliação de Desempenho

A figura 1 apresenta os tempos de execução para as cinco plataformas de testes e para diferentes tamanhos de matrizes. Os valores obtidos pelo *NumPy* podem ser considerados como valores de referência para a MM, dado que é uma biblioteca consolidada no mercado e apresenta diversas otimizações.

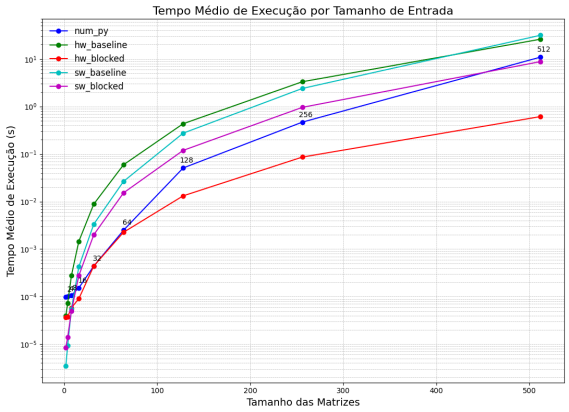

Figura 1. Gráfico em escala logarítmica da média dos tempos de execução por tamanho de matriz para diferentes plataformas

Os tempos de execução em software (*baseline* e *blocked*) apresentaram melhor desempenho em relação ao hardware até matrizes 8x8. A partir de 16x16, o HW *blocked* mostrou tempos de execução entre 10 a 20 vezes menores que o HW *baseline* e o SW *baseline* e *blocked*, ao passo que os tempos de HW *baseline* aumentam e passam a ter o menor desempenho comparado aos demais. Ao mesmo tempo, todas plataformas e tamanhos apresentam tempos de execução maiores quando comparados aos obtidos pelo *NumPy*, exceto o HW *blocked* que apresentou tempos até 20 vezes menores para matrizes 128x128 ou maiores.

3.4. Análise de Recursos Utilizados

A tabela 1 mostra o uso percentual de recursos da placa *PYNQ-Z2* ao sintetizar os algoritmos *baseline* e *blocked* com diferentes tamanhos de matrizes de entrada, respectivamente. Os recursos utilizados são os seguintes:

1. *LUT*: tabela de mapeamento entre entradas e saídas para operações lógicas;
2. *LUTRAM*: variação das LUTs com capacidade de armazenamento, atuando como pequenas RAMs;
3. *FF*: circuito sequencial que armazena dados até um sinal de controle (e.g., clock);
4. *BRAM*: módulos de RAMs capazes de integrar com diferentes layouts e interfaces;
5. *DSP*: unidades para operações aritméticas e lógicas (e.g., multiplicação, adição);
6. *BUFG*: buffer global que distribui sinais de clock pela FPGA.

Durante os experimentos, constatou-se que o consumo de recursos do algoritmo *baseline* é independente do tamanho da matriz de entrada. O uso de BUFG permanece constante em ambos os algoritmos. Por outro lado, o algoritmo *blocked* utiliza mais recursos conforme as matrizes de entrada crescem, tendo um aumento notável a partir de 256x256. Nota-se também que para 256x256, o uso de DSP atinge 100%, de modo que

Recurso	Disponibilidade	Blocked									Baseline
		2x2	4x4	8x8	16x16	32x32	64x64	128x128	256x256	512x512	
LUT	53200	4,011	4,160	4,479	4,812	5,727	7,466	8,594	18,009	59,633	4,523
LUTRAM	17400	1,132	1,178	1,282	2,609	4,161	7,103	7,167	12,966	17,603	1,109
FF	106400	2,758	3,058	3,448	3,251	4,004	5,213	7,552	15,734	36,989	3,414
BRAM	140	0,714	0,714	0,714	1,071	1,071	1,071	1,071	1,071	100	0,714
DSP	220	0,909	1,818	3,636	7,273	14,545	29,091	58,182	100	100	1,364
BUFG	32	3,125	3,125	3,125	3,125	3,125	3,125	3,125	3,125	3,125	3,125

Tabela 1. Utilização percentual dos recursos do algoritmo baseline e em blocos para diferentes tamanhos de matrizes

para 512x512 o consumo de DSP permanece em 100% e o consumo de BRAM passa de 1,071% para 100%. Não foi possível sintetizar o circuito para a MM em blocos para matrizes maiores que 512x512.

4. Conclusão

Neste artigo, foram obtidos e analisados dados dos tempos de execução para dois algoritmos de multiplicação de matrizes, padrão (*baseline*) e em blocos (*blocked*), considerando suas versões em software e em hardware, sintetizadas por meio de HLS. Também foi estudado o consumo de recursos de ambos os algoritmos ao sintetizá-los na placa *PYNQ-Z2*.

A multiplicação de matrizes HW *blocked* apresentou melhor desempenho sobre todas as outras versões para matrizes a partir de 16x16. Também foi visto que o uso de recursos é constante para o HW *baseline*, ao passo que a alocação de recursos cresce com o tamanho da matriz para o HW *blocked*.

Para trabalhos futuros, podem ser feitos: (i) estudo da influência do tamanho dos blocos para um tamanho fixo de matriz na MM em blocos; (ii) estudo da estrutura do circuito RTL gerado pelo Vitis para outros algoritmos, por exemplo, MDC (máximo divisor comum) e raiz quadrada inteira.

Referências

Kastner, R., Matai, J., and Neuendorffer, S. (2018). Parallel Programming for FPGAs. *ArXiv e-prints*.

Leon-Vega, L. G. and Castro-Godinez, J. (2023). Generic accuracy configurable matrix multiplication-addition accelerator using hls. *Proceedings - 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops Volume, DSN-W 2023*, pages 171–174.

Lu, J. and Chen, W. (2024). High-level-synthesis design flow on zynq. Disponível em: `https://github.com/Xilinx/xup_high_level_synthesis_design_flow/tree/main`.

Meng, X., Zhuang, W., Qin, Z., Yu, L., and Hou, G. (2022). The design and implementation of complex float matrix multiplication operation based on high-level synthesis. *Proceedings - 2022 International Conference on Computing, Robotics and System Sciences, ICRSS 2022*, pages 40–44.

Skalicky, S., Wood, C., Lukowiak, M., and Ryan, M. (2013). High level synthesis: Where are we? a case study on matrix multiplication. *2013 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2013*.