Comparando orientação a dados e orientação a objetos em análise de *timing* estática

Sheiny Fabre Almeida, Bernardo Ferrari Mendonça, Laércio Lima Pilla, José Luís Almada Güntzel

Departamento de Informática e Estatística – Universidade Federal de Santa Catarina
Florianópolis – SC – Brasil

{sheiny.fabre, bernardo.mendonca}@grad.ufsc.br, laercio.pilla@ufsc.br

Resumo. *Este artigo apresenta uma comparação entre a abordagem tradicional de programação orientação a objetos com a programação orientada a dados. Os experimentos foram realizados no contexto da Análise de Timing Estática, um método utilizado para estimar o atraso de um Circuito Integrado. Os resultados incluem os tempos de execução com relação a estrutura de dados, processamento e paralelismo.*

1. Introdução

Os Circuitos Integrados (CIs) constituem o núcleo de qualquer equipamento eletrônico contemporâneo. Durante o projeto de um CI, diversas otimizações iterativas são realizadas para satisfazer as restrições de projeto, como por exemplo: atraso, área e potência. Para guiar tais otimizações, informações de atraso são necessárias.

A Análise de *Timing* Estática (ATE) é um método utilizado para computar o atraso esperado de um circuito digital sem realizar qualquer tipo de simulação [Kahng et al. 2011, p. 222]. Foi verificado que mais da metade do tempo necessário para realizar a ATE é utilizado para calcular o atraso dos elementos do circuito. O atraso dos elementos do circuito é calculado realizando uma interpolação sobre uma tabela. Porém, ao realizar a interpolação, apenas parte da tabela é utilizada assim, ficando parte da *cache* ocupada por dados irrelevantes no instante.

Tendo em vista esses aspectos, este artigo propõe uma comparação entre a Orientação a Objetos - *Object Oriented Design* (OaO) e a Orientação a Dados - *Data Oriented Design* (OaD) para explorar melhor o acesso às estruturas de dados.

2. Cálculo do Atraso das Portas

Por questões de complexidade, o projeto de um CI adota um fluxo de projeto conhecido como *standard cell*. Tal fluxo utiliza uma biblioteca de células, onde estas possuem informações obtidas através de simulação no nível elétrico [Kahng et al. 2011, p. 13].

Para calcular o atraso de uma porta lógica, é feita uma interpolação sobre uma tabela, fornecida pela biblioteca. A interpolação é feita em função do tempo de transição do sinal na entrada da porta (*slew*) e da capacitância associada à saída da porta [Bhasker and Chadha 2009, p. 47]. Esta relação está ilustrada pela Figura 1.

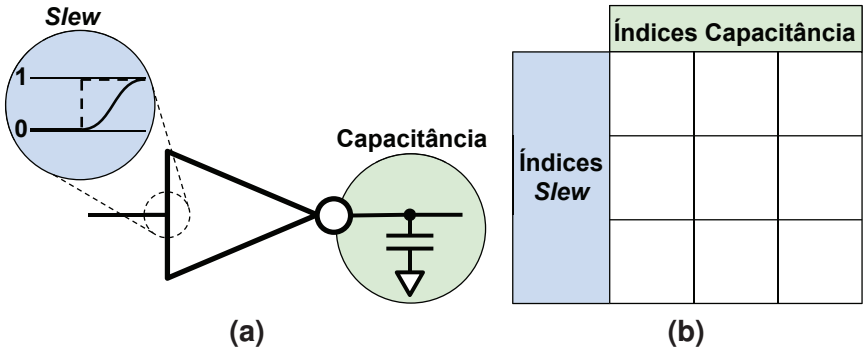

Figura 1. (a) À esquerda está ilustrado o *slew* na entrada da porta, e à direita a capacitância associada a saída da porta. (b) Tabela onde as linhas são indexadas pelo *slew* e as colunas pela capacitância.

3. Orientação a Dados

O paradigma de OaD tem como objetivo o alto desempenho, priorizando as estruturas de dados e a forma como estas são operadas. Ele se difere da abordagem tradicional, OaO, a qual tem como objetivo modelar objetos do mundo real e suas relações.

3.1. Estrutura de Dados

A diferença em relação à estrutura de dados está ilustrada pela Figura 2, onde na esquerda está ilustrada a abordagem orientada a objetos, e à direita, a orientação a dados.

- Na versão orientada a objetos 2a: cada tipo de porta contém uma tabela com os índices e valores para interpolação.
- Na versão orientada a dados 2b: cada tipo de porta é armazenado em um vetor, onde este faz referência a vetores de diferentes características que estão contíguos na memória.

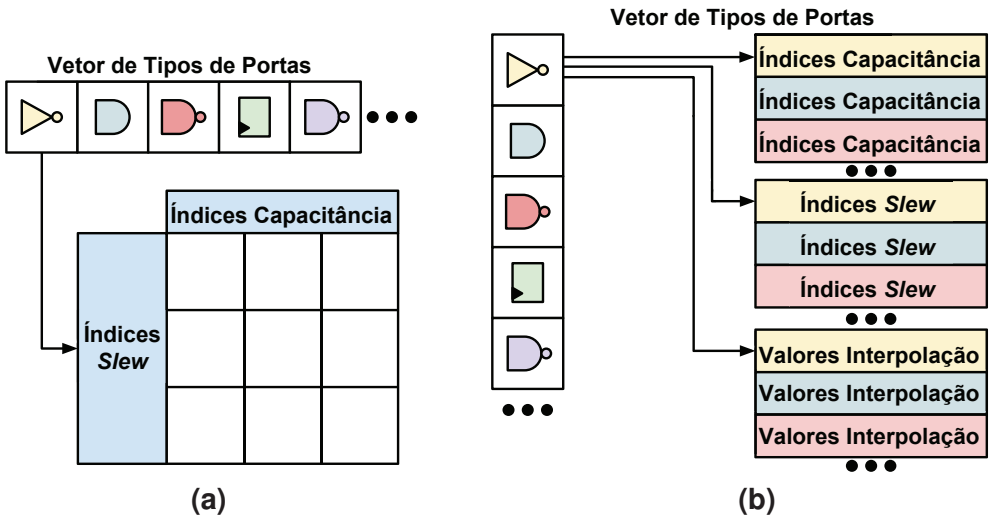

Figura 2. Ilustração da estrutura de dados orientada a objetos (a) e orientada a dados (b).

3.2. Processamento dos Dados

O modelo computacional utilizado na ATE é um grafo direcionado acíclico dividido em níveis, onde cada nível é formado por um conjunto de portas e estas precisam ter o atraso calculado [Huang and Wong 2015]. Os Algoritmos 1 e 2 ilustram o cálculo dos atrasos, para as diferentes abordagens, OaO e OaD respectivamente.

Seja P o conjunto de portas e p_i uma porta $\in P$.

Seja D o conjunto de atrasos, onde $d_i \in D$ e d_i é o atraso da porta p_i.

Algoritmo 1: Processamento por Portas

```
   Entrada: Conj. de portas P
   Saída  : Conj. de atrasos D
1  para p_i ∈ P faça
2  |  Buscar índice i (slew)
3  |  Buscar índice j (cap.)
4  |  Realizar interpolação (i, j)
5  fim
```

Algoritmo 2: Processamento por Característica

```
   Entrada: Conj. de portas P
   Saída  : Conj. de atrasos D
1  para p_i ∈ P faça
2  |  Buscar índice i (slew)
3  fim
4  para p_i ∈ P faça
5  |  Buscar índice j (cap.)
6  fim
7  para p_i ∈ P faça
8  |  Realizar interpolação(i, j)
9  fim
```

No Algoritmo 1, **cada porta é processada em uma única iteração**: primeiro são buscados os índices de *slew* (i), em seguida capacitância (j) e então é realizada a interpolação.

No Algoritmo 2, **uma característica é processada por iteração**. Para cada porta são buscados: no primeiro laço (linha 1), os índices de *slew* (i), no segundo laço (linha 4), os índices de capacitância (j). E então na última iteração (linha 7), todas as portas são processadas, realizando a interpolação.

4. Metodologia da Avaliação de Desempenho

Os experimentos foram realizados para analisar o comportamento das diferentes abordagens com relação ao acesso a memória, variando os parâmetros:

- **Estrutura de dados:** Utilizando a orientação a objetos ou a orientação a dados.
- **Processamento dos dados (Subseção 3.2):** Processando todos os dados em uma única iteração ou em várias iterações divididas por diferentes características.
- **Ordenamento das portas:** O conjunto P apresentado na Subseção 3.2, pode ser ordenado por tipo de porta para explorar a localidade temporal da *cache*, pois uma tabela será referenciada novamente por uma porta do mesmo tipo.
- **Paralelismo:** Executando em paralelo (OpenMP), ou sequencialmente.

Ambiente de testes: processador AMD Phenom™ II X6 1090T (3.2 GHz) 8 GB de memória, sist. op. Linux Mint 17.3 Rosa 64 bits e compilador gcc versão 5.4.1. Para reduzir o ruído nos experimentos, foi utilizada a biblioteca *Portable Hardware Locality* (hwloc) v1.11.2, onde as *threads* foram mapeadas aos núcleos.[1]

[1]Código disponível em: `https://github.com/sheiny/ophidian/tree/STAwip/test`

Os experimentos foram realizados com um nível da ATE contendo 4096 portas, e uma biblioteca de células com 211 tipos diferentes de portas lógicas. Para cada experimento foram feitas 30 execuções.

5. Resultados

De acordo com a Tabela 1, a granularidade deste problema não justifica o uso de paralelismo. Com relação ao ordenamento das portas por tipo, para todos os casos, o sobrecusto da ordenação foi em média de 287 μs, o que acaba invalidando tal otimização.

Tabela 1. Tempo em μs para a variação dos parâmetros (Seção 4).

Estrutura de dados	Modo	Ord. das Portas	Paralelo (6 threads esc. estático)	Tempo médio (μs)	Desvio padrão (σ)	Intervalo de conf. (95%) (μs)
OaD	OaD	Sim	Sim	1079	509	1079 ± 182
OaD	OaD	Sim	Não	901	554	901 ± 198
OaD	OaD	Não	Sim	842	374	842 ± 134
OaD	OaD	Não	Não	687	377	687 ± 135
OaD	OaO	Sim	Sim	1115	426	1115 ± 152
OaD	OaO	Sim	Não	866	461	866 ± 164
OaD	OaO	Não	Sim	943	420	943 ± 150
OaD	OaO	Não	Não	663	218	663 ± 78
OaO	OaD	Sim	Sim	744	270	744 ± 96
OaO	OaD	Sim	Não	549	293	549 ± 104
OaO	OaD	Não	Sim	705	296	705 ± 106
OaO	OaD	Não	Não	565	256	565 ± 91
OaO	OaO	Sim	Sim	774	275	774 ± 98
OaO	OaO	Sim	Não	532	259	532 ± 93
OaO	OaO	Não	Sim	724	336	724 ± 120
OaO	OaO	Não	Não	573	235	573 ± 84

6. Conclusões

O principal motivo para a orientação a dados não ter obtido um bom desempenho foi que para realizar as operações, múltiplos vetores em diferentes regiões da memória precisam ser acessados, o que acaba comprometendo a localidade espacial da *cache*. Portanto para este experimento, a orientação a objetos parece ser mais adequada, pois um objeto contém todas informações necessárias para realizar uma dada tarefa, sendo estas localizadas em uma única região da memória.

Referências

Bhasker, J. and Chadha, R. (2009). *Static timing analysis for nanometer designs: a practical approach*. Springer Science & Business Media.

Huang, T.-W. and Wong, M. D. (2015). Opentimer: A high-performance timing analysis tool. In *Proceedings of the IEEE/ACM ICCAD*, pages 895–902. IEEE Press.

Kahng, A. B., Lienig, J., Markov, I. L., and Hu, J. (2011). *VLSI Physical Design: From Graph Partitioning to Timing Closure*. Springer Publishing, Incorporated, 1st edition.