# Propostas de Otimização de uma Implementação do Algoritmo de Análise Diferencial de Potência

**Rodrigo Bazo[*], Diego Poleto[†], Gerson Geraldo H. Cavalheiro, Rafael Soares**

[1]Programa de Pós-Graduação em Computação
Universidade Federal de Pelotas
Pelotas – RS – Brasil

rbazo@unisinos.br
{diegopoleto, gerson.cavalheiro,rafael.soares}@inf.ufpel.edu.br

***Resumo.*** *Ataques por canais laterais exploram vulnerabilidades de um sistema criptográfico por meio do monitoramento de grandezas físicas tais como consumo de energia e tempo de execução. Estes ataques exigem grande poder computacional por atuarem em uma grande quantidade de dados e por explorar diferentes variações do ataque durante a execução. Neste trabalho é apresentada uma implementação inicial de Análise Diferencial de Potência (DPA) e são propostos quatro modos de otimização a fim de explorar seu paralelismo e reduzir o tempo de execução. Os resultados obtidos em uma arquitetura multiprocessada mostram ganhos significativos de desempenho, reduzindo em até 3000 vezes o tempo de execução do algoritmo DPA.*

## 1. Introdução

A Internet disponibilizou à população facilidades na realização de transações como compras de produtos, pagamento de contas, operações bancárias e semelhantes. A ubiquidade desejada no acesso a estes recursos fez com que diferentes equipamentos, como terminais bancários para auto atendimento e máquinas leitoras de cartão, sejam disponibilizados em locais públicos. Com a utilização em massa destes serviços e equipamentos, há uma grande preocupação com o sigilo dos dados, sejam os dados relativos às transações efetuadas, sejam os dados relativos às senhas de acesso.

Para garantir sigilo dos dados, são utilizados protocolos de comunicação e criptografia com a finalidade de ocultar o conteúdo de uma determinada mensagem quando em trânsito entre sua origem e seu destino. Neste caso, o sigilo das mensagens fica condicionado a chave criptográfica, uma pequena palavra que deve ser conhecida apenas pelos entes comunicantes com chave simétricas. Apesar disso, existe a criptoanálise, ciência dedicada a encontrar vulnerabilidade em algoritmos criptográficos a fim de extrair informações sigilosas de sistemas que usam algoritmos criptográficos.

Os algoritmos criptográficos evoluíram significantemente nos últimos anos, muito em função do fato de os algoritmos serem públicos e da intensa atividade de criptoanálise. Considerando isto, existe um grande esforço da comunidade científica para identificar as vulnerabilidades de sistemas criptográficos a fim de encontrar soluções que possam resistir às ameaças provocadas por ataques.

[*]Bolsista IC Capes
[†]Bolsista IC Capes

De acordo com [Waddle and Wagner 2004], existem dois tipos de ataques aos sistemas criptográficos: aqueles que procuram falhas nos próprios algoritmos de criptografia e aqueles que exploram as vulnerabilidades físicas da implementação do sistema. Neste segundo caso, o ataque é realizado por meio do monitoramento do dispositivo físico que está executando o algoritmo criptográfico. Esta abordagem é conhecida como Ataque por Canais Laterais (do inglês, *Side-Channel Attacks* – SCA) [Kocher 1996].

A Análise Diferencial de Potência (*Differential Power Analysis* – DPA) é um ataque do tipo SCA que, por meio de análises estatísticas, estabelece uma dependência entre os dados processados por algoritmos criptográficos tais como (*Data Encryption Standard* - DES), (*Advanced Encryption Standard* - AES) e o consumo de energia do circuito que o executa [Kocher et al. 1999]. Ataques DPA são principalmente aplicados a sistemas criptográficos que utilizam o algoritmo DES ou sua versão moderna, o AES . O grande desafio é aumentar o nível de segurança destes sistemas, certificando-se a robustez de seus métodos de segurança ao vazamento de informações por meio de sua dissipação de potência antes de sua disponibilização ao mercado consumidor.

DPA é um tipo de análise que demanda alto poder de processamento por manipular uma grande quantidade de traços contendo o consumo de corrente do dispositivo atacado e consequentemente revelando a potência dissipada durante o processamento. Por se tratar de um método estatístico, sendo maior a quantidade de traços de medição de consumo energético, melhor será a qualidade da análise estatística e, portanto, melhor o resultado da criptoanálise. O estudo de caso realizado neste artigo explora a otimização deste algoritmo do ataque DPA explorando sua paralelização em uma arquitetura multiprocessada a fim de dinamizar as pesquisas relacionadas à área de criptoanálise pela redução do tempo necessário à obtenção de resultados de análises. O ponto de partida se dá pelo desenvolvimento de um código em Matlab para sua implementação e, na sequência, sua tradução para C++ e sucessivas otimizações de código visando desempenho por meio de paralelização em diferentes níveis.

O restante deste artigo está organizado como segue. A Seção 2 descreve o funcionamento do algoritmo DES, como processo usado para encriptar e decriptar traços da análise de potência, bem como alguns tipos de ataques existente, destacando aquele usado como estudo de caso. A Seção 3 fornece embasamento sobre o funcionamento dos ataques por canais laterais em meio a implementação das funções *DPA* e *KeySearch*, que se destacam pelo alto custo computacional, sendo assim alvos para otimização. A Seção 4 refere-se a trabalhos relacionados, menciona implementações que se destacam por explorar o paralelismo do algoritmo DPA e o uso de GPUs para melhoria do tempo de execução. Na seção 5 estão presentes as propostas de otimização do trabalho, assim como os resultados obtidos em cada etapa. As conclusões são apresentadas na Seção 6.

## 2. Algoritmo de criptografia DES

Os algoritmos DES e AES são *simétricos* por utilizarem a mesma chave criptográfica para cifrar e decifrar os dados. O algoritmo DES usado neste trabalho como estudo de caso, é composto de três etapas: permutação inicial, 16 rodadas de execução de uma função aplicada sobre a mensagem de entrada e chave criptográfica e uma permutação final. Os dados são efetivamente processados em blocos de 56 bits, sendo 8 bits de paridade. Os ataques geralmente ocorrem nas SBOXs, pois é onde a chave e os dados estão sendo

encriptados. Na Figura 1 a estrutura de uma rodada do algoritmo DES é representada, os passos das rodadas são descritos a seguir de acordo com [Stallings 2008]:.

1. Um bloco de dados de entrada de 64 bits, onde 8 bits são paridade, reduzindo-se a um bloco de 56 bits de informação propriamente dita; Em seguida, este bloco é dividido em dois blocos;
2. Paralelo a este fluxo, a chave criptográfica de 64 bits sofre uma permutação inicial e redução para 48 bits chamada *Round Key*, onda a cada rodada a chave sofre uma operação de escalonamento;
3. Aplicam-se operações lógicas OU-Exclusivo (do inglês, *Exclusive Or*) utilizando as *Round Keys*, que são calculadas a partir da chave do usuário;
4. Os resultados são divididos e encaminhados para 8 Sboxes (do inglês, *Substitution Boxes*); Novamente é aplicada a operação OU-Exclusivo; Ao fim da rodada é realizada a união dos blocos de dados para formar a saída.

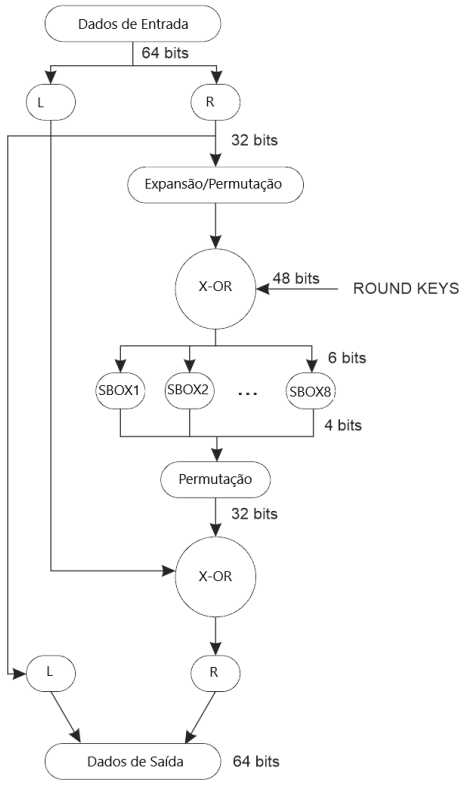


**Figura 1. Fluxograma de execução das rodadas do DES.**

Ao fim das 16 rodadas é executada uma permutação final e tem-se então o dado cifrado. Na Figura 2, está ilustrado o comportamento de um traço de consumo de um dado, sendo processado pelo algoritmo DES. Na figura é destacada a permutação inicial, as 16 rodadas do algoritmo e a permutação final.

## 3. Ataques por Canais Laterais

Os ataques por canais laterais visam explorar informações contidas em canais laterais, esses canais podendo ser, por exemplo, consumo de energia, tempo de execução e radiação

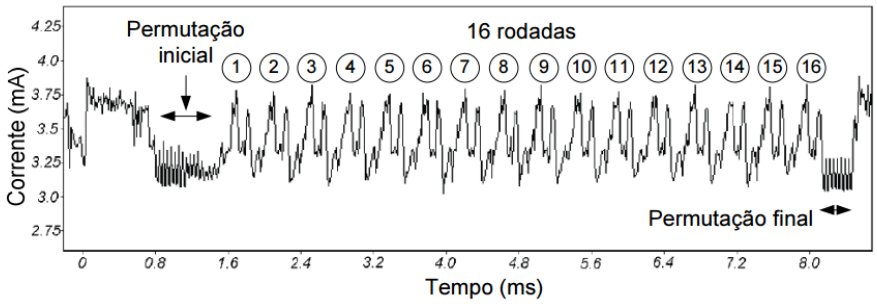


**Figura 2. Modelo de traço de consumo de um processamento do DES .**

eletromagnética. Os ataques por consumo de potência visam descobrir o dado processado analisando o consumo de energia do dispositivo executando o algoritmo criptográfico. Para isso é necessário o uso de um equipamento para aquisição do consumo de energia do dispositivo alvo do ataque, normalmente usa-se um osciloscópio para tal propósito conforme ilustrado na Figura 3. Os dispositivos atacados são normalmente smartcards, porém qualquer dispositivo poderia ser atacado como por exemplo um microcontrolador, FPGA ou um circuito integrado de aplicação específica (do inglês, Application Specific Integrated Circuit - ASIC).

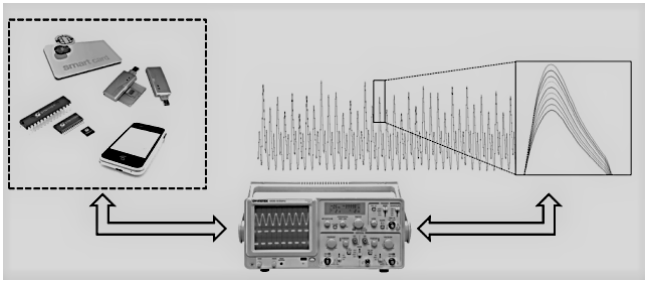

**Figura 3. Alguns dispositivos que podem ser sujeitos à ataques por DPA.**

### 3.1. Implementação do algoritmo DPA

O ataque visa analisar uma função alvo do algoritmo DES que relacione dado de entrada e chave criptográfica a fim de descobrir a chave criptográfica a partir dos traços de consumo de energia. No algoritmo DES as funções alvo normalmente são as SBOXs. Após obtidos os traços de consumo para um conjunto com diferentes dados de entrada e mesma chave criptográfica a etapa seguinte é realizar a análise propriamente dita. Para isso aplica-se uma função de seleção ao conjunto de traços adquiridos. Esta função divide o conjunto de traços disponíveis em dois grupos, a fim de identificar para cada hipótese de chave se o bit alvo do ataque é zero ou um.

O pressuposto do ataque é que o consumo de energia em circuitos digitais implementados com tecnologia CMOS difere para transições $0 \rightarrow 1$ e $1 \rightarrow 0$. Assim, o ataque classifica traços de consumo em dois grupos onde em um dado instante do tempo de processamento ocorre uma transição $0 \rightarrow 1$ e no outro, uma transição $1 \rightarrow 0$. A divisão correta de traços garante que, com o cálculo do traço médio de cada grupo, o consumo alvo do ataque seja evidenciado em um dado instante. Assim, a diferença entre os traços

médios obtidos produzirá uma diferença significativa entre os traços no dado instante alvo do ataque. A curva obtida pela diferença das médias de cada grupo é chamada de curva hipótese de chave. Uma curva para cada hipótese de chave é gerada durante a análise. A curva com maior amplitude no dado instante de processamento é considerada a curva de maior probabilidade de chave correta.

A função de seleção dos traços pode usar diferentes estratégias para desempenhar o ataque. Analisar individualmente os bits de saída de cada SBOX é estratégia mais comum. Neste caso é possível realizar ataques diferentes para cada um dos 4 bits de saída das SBOXs. Outra possibilidade de função seleção é analisar a soma dos 4 bits da SBOX (SUM). Uma terceira estratégia é avaliar o Peso Hamming (HW) da saída da SBOX. Além disso é possível executar um ataque que testa todas as estratégias de função seleção disponíveis (ALL). Tal como DPA, outras estratégias alternativas de ataques foram propostas, uma delas é CPA onde um modelo de consumo de energia é aplicado aos traços de consumo a fim de evitar influências de ruídos presentes nos traços e tornar o ataque mais eficiente [Brier et al. 2004].

A quantidade de traços envolvida no ataque impacta diretamente no custo computacional despendido, assim como a quantidade de amostras contidas nos traços. A implementação do algoritmo aninha alguns laços para realizar o ataque o que resulta no longo esforço computacional sem explorar o paralelismo do algoritmo. O ataque deve analisar todo o conjunto de traços que normalmente é superior a 50 mil. O algoritmo DES possui 8 SBOXs, logo são realizados 8 ataques para a obtenção de todos os segmentos da chave criptográfica. No entanto, cada SBOX pode ser atacada de diferentes maneiras conforme revisado, um ataque a cada bit, SUM, HW e CPA ou ainda ALL, a execução de todos eles. Isto é replicado para cada SBOX. Cada ataque gera uma curva hipótese para cada uma das 64 hipóteses de chave para a SBOX por meio da função *KeySearch*, uma das funções que mais demandam processamento do algoritmo, justamente por analisar e organizar as hipótese de chave para todos os tipos de ataque realizados. Logo, o uso do paralelismo juntamente com um acelerador gráfico deve reduzir o tempo de execução do algoritmo conforme [Swamy et al. 2014].

### 3.2. Função KeySearch

A função *KeySearch* mostrada na Figura 4 é responsável por percorrer a matrizes N e identificar as hipóteses de chave correta. No início da execução são inicializadas algumas variáveis como por exemplo a `matrixMax`, que armazena todos os valores máximos de cada curva obtida pela diferença das médias contida na matriz N a ser pesquisada.

Esta função pode usar dois métodos de busca da melhor hipótese de chave. Um método usa a maior amplitude da curva diferencial, enquanto o outro avalia a maior área sobre um intervalo pré-definido. Neste último caso, usa-se uma função que retorna a integração trapezoidal sob a curva. Ambos métodos são eficazes para a definição da melhor hipótese de chave na matriz N.

## 4. Trabalhos Relacionados

Em [Bartkewitz and Lemke-Rust 2011] é proposta uma implementação do ataque DPA dedicada para a execução em unidades de processamento de gráfico (GPUs). A implementação utiliza recursos avançados oferecidos pelo *CUDA Framework* para minimizar o

```
1: INITVARS()
2: for each keyHypothesis k do
3:     if method == maxPeak then
4:         Calculate matrixDPA maximum peak
5:     end if
6:     if method == maxIntegral then
7:         Calculate matrixDPA maximum trapezoidal integral
8:     end if
9: end for
10: keyGuessValue = MAXIMUM(matrixMax)
11: keyGuess = maximumValuePosition
12: matrixValuesIndexes = DESCENDINGSORT(matrixMax)
13: for each keyHypothesis k do
14:     if matrixValuesIndexes.Indexes(k) == roundKey then
15:         rankRealKey = k
16:         break
17:     end if
18: end for
19: keyGuessValue2 = GETSECONDMAXIMUM(matrixMax)
20: margin = CEIL(100 - ((keyGuessValue - keyGuessValue2)/keyGuessValue))
21: abscissa = matrixPoint[keyGuess] + xMin
```

**Figura 4. Pseudocódigo genérico da função keySearch.**

tempo de execução com base no coeficiente de correlação de Pearson. O modo de calcular este coeficiente poderia servir como uma referência para processamento de alto desempenho, pois como mostrado pelos autores é possível analisar um milhão de traços contendo 20 mil amostras em menos de dois minutos.

Em [Swamy et al. 2014], os autores propuseram a implementação de uma plataforma usando código aberto para execução de ataques a canais laterais, para avaliar a segurança de uma gama de dispositivos em relação a ataques SCA. Para isso foi desenvolvida uma estrutura de processamento paralelo com objetivo de acelerar cálculos intensivos explorando o paralelismo em nível de dados da aplicação, algo que é inerente aos algoritmos de ataque SCA. Para execução da implementação proposta utilizaram uma unidade de processamento gráfico (GPU), para acelerar a extração de chaves criptográficas.

No trabalho de [Brier et al. 2004] é proposta uma especialização de DPA, chamada de Análise por Correlação de Potência (*Correlation Power Analysis*). Esta análise utiliza o modelo de potência distância Hamming. Esta proposta visa reduzir o problema dos picos fantasmas em DPA. Atualmente existem alguns trabalhos que propõem modelos para acelerar a CPA, como no trabalho de [Gamaarachchi et al. 2014] onde os autores propõem a execução deste problema com GPUs utilizando CUDA.

Conforme revisado, os trabalhos encontrados na literatura exploram o paralelismo dos algoritmos de ataque DPA e execução em GPUs conhecidas por seu elevado número de elementos de processamento. No entanto, o uso de GPUs requer implementação dedicada com o uso de uma interface de programação para o uso dos recursos. Por outro lado, o uso de plataformas *multicore*, vários núcleos de processamento com possibilidade de execução de múltiplas *threads* não possui o mesmo desempenho em relação as executadas em GPUs, mas possuem a vantagem de manter a mesma estrutura do algoritmo sem a necessidade do uso de interfaces de programação específicas para cada tecnologia.

Neste trabalho é explorado o paralelismo de uma implementação do ataque DPA usando plataformas *multicore* para sua execução.

## 5. Propostas de Otimização da Implementação

Nesta Seção são apresentadas quatro propostas de otimizações aplicadas ao código que implementa o ataque DPA. A implementação disponível e validada em [Lomné et al. 2009] encontra-se codificada para o Matlab devido aos seus grandes recursos para operações e manipulações de matrizes. Entretanto, a execução de programas no Matlab é interpretada, sendo executada em uma máquina virtual o que não favorece implementações que exigem alto desempenho. Desta forma, otimizações são propostas e análises de desempenho são executadas para avaliar o tempo de execução das mesmas. As avaliações de desempenho consideram quatro estratégias da função de seleção de traços: Sum, HW, CPA e ALL, além dos seguintes parâmetros:

- Traço de consumo com 1.500 amostras;
- Métodos de busca da chave: maxPeak e maxIntegral;
- Modelo de consumo: HD (*Hamming Distance*).

Os experimentos foram realizados em um PC com a seguinte configuração: Processador Core i5 4690 com 4 cores físicos, 8 Gb de Memória RAM, executando o sistema operacional Ubuntu 14.04 e utilizando o compilador GCC versão 4.9. Os testes são realizados sobre um conjunto de 100.000 traços de consumo de energia provenientes de um sistema criptográfico alvo do ataque. Ataques executados sobre um conjunto desta ordem dispendem um tempo significativo de processamento. Um teste inicial mede o tempo de execução do ataque sobre o conjunto disponível. Em seguida, executa-se o mesmo ataque sobre um subconjunto de 5.000 traços, onde a cada 100 traços processados é medido e armazenado o tempo de execução. Ao final, calcula-se o tempo médio de execução de 100 traços e estima-se o tempo de execução para 100.000 traços. O tempo estimado é aproximadamente o mesmo obtido no teste inicial sobre todo o conjunto, garantindo a possibilidade das avaliações de desempenho apenas sobre um subconjunto dos traços e reduzindo os tempos de processamento.

### 5.1. Tradução do Algoritmo

O código desenvolvido em C++ respeita a estrutura do algoritmo apresentado em [Lomné et al. 2009]. No entanto, como a linguagem C++ não oferece os mesmos recursos matemáticos e funções específicas para cálculo de integral e manipulação de matrizes, algumas funções foram implementadas. Como esperado, o ganho de desempenho obtido nesta etapa de otimização foi significativo conforme mostrado na Figura 5. Deve-se observar que a estrutura inicial do algoritmo foi preservada a fim de avaliar-se apenas a mudança da linguagem de programação nesta primeira otimização proposta.

Os resultados obtidos nesta etapa de otimização destacam que apesar de oferecer diversas facilidades para implementação de operações com matriz, o Matlab não é uma solução interessante para aplicações que exigem alto desempenho. Por ser uma linguagem interpretada e oferecer um alto nível de abstração, o seu desempenho é inferior a implementações em linguagem compiladas tais com a linguagens como C++ gerando um código executável pelo processador e favorecendo aplicações de alto desempenho.

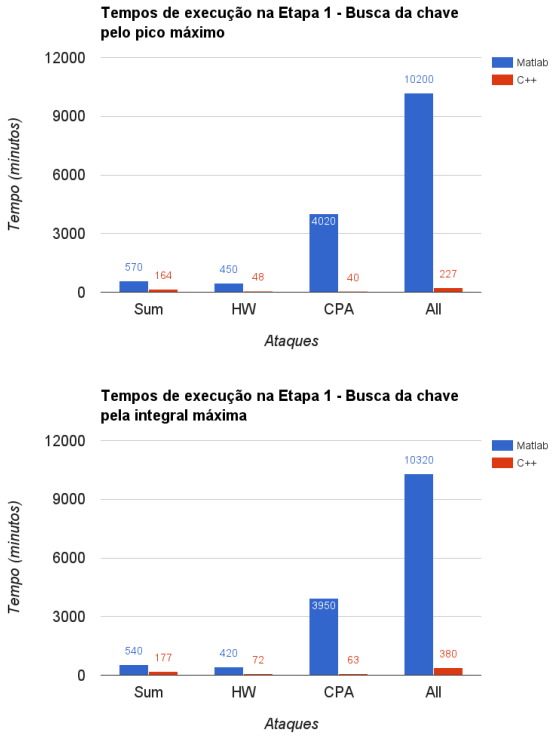


**Figura 5. Comparação dos tempos de execução entre os algoritmos em C++ e em Matlab na Etapa 1.**

### 5.2. Planificação de Matrizes

Analisando a execução do algoritmo observa-se que a parte de maior custo computacional é a exigência por percorrer as matrizes tridimensionais iterando por meio das colunas das matrizes e não em sua profundidade, ou seja, há um laço de iteração `for i`, seguido de um `for k` e um `for j`. O principal problema ocorre pelo fato do termo que está iterando ser o termo das colunas, ou seja, se tal vetor da matriz está localizado em uma posição de memória não carregada na memória cache ocorre um `cache mis` gerando um tempo de atualização da cache. Esse processo pode se repetir inúmeras vezes comprometendo o desempenho da execução do algoritmo. Para amenizar o custo da busca de tais vetores em memória propõe-se a planificação destas matrizes tridimensionais para vetores unidimensionais como mostra na Figura 6.

Nesta matriz pode-se notar que seus índices foram remapeados para um vetor que possui tamanho `numKeyHypothesis * numPontos * numSbox`. Os índices `kps`, `kPS` e `KPS`, onde k é a linha, p são as colunas e s é a profundidade, indicam respectivamente as posições `[0][0][0]`, `[0][numPontos][0]` e `[numKeyHypothesis][numPontos][numSbox]` estão destacados no vetor para indicar as posições em que foram remapeados. Para realizar o controle dos índices desta matriz tridimensional, dentro de um laço de iteração for i, j e k, foi utilizada a equação `[i * (largura * profundidade) + j * (profundidade) + k]`. Com esta equação foi mantida a coerência dos índices de cada dado da matriz original.

A Figura 7 mostra que o tempo de execução da busca pelo pico máximo não

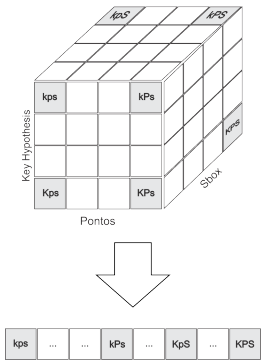


**Figura 6. Planificação de matriz tridimensional em um vetor.**

foi reduzido apenas no CPA. O Sum, ataque com maior consumo de memória dentre os avaliados, teve uma redução em torno de 39 minutos em seu tempo de execução. O HW teve uma redução de aproximadamente 10% (≈5 min) em relação a etapa anterior. Ao utilizar todos os ataques, o tempo de execução foi reduzido em torno de 25 minutos. Na busca pela integral máxima o comportamento se manteve, apesar de o ganho no Sum ter sido um pouco menor em relação a busca pelo pico máximo. Os demais resultados mantiveram a proporção de seus ganhos, onde o ALL teve uma redução em torno de 30 minutos em sua execução. Pode-se observar que o CPA foi o único ataque que não obteve ganho de desempenho nesta etapa, isso se deve pelo fato de este ataque não possuir matrizes intermediárias como os demais.

### 5.3. Diretivas SIMD

Após a planificação das matrizes, foi utilizado o OpenMP para fazer a implementação de diretivas SIMD neste algoritmo. Estas diretivas informam o compilador que determinado laço pode ser executado utilizando instruções do tipo SIMD (*Single Instruction, Multiple Data*). Estas instruções fazem com que diversas computações sejam executadas paralelamente, porém somente uma instrução por vez. Este tipo de diretiva também é conhecido como paralelismo de dados.

Em cada laço que percorre as matrizes intermediárias foram implementadas as diretivas `#pragma omp simd`. Entretanto, os resultados nesta etapa foram os que obtiveram menor ganho de desempenho em relação as demais etapas de otimização propostas, conforme mostrado na Tabela 1.

Em ambos os métodos de busca se observou um pequeno ganho de desempenho nos mesmos ataques da etapa anterior: Sum, HW e ALL. Novamente o CPA manteve o desempenho da etapa anterior por não ter matrizes intermediárias. Na Tabela 1 observam-se os ganhos de desempenho obtidos nesta etapa foram pequenos.

**Tabela 1. Speedups da vetorização do código na Etapa 3.**

| | Sum | HW | CPA | ALL |
|---|---|---|---|---|
| maxPeak | 1.03 | 1.04 | 1 | 1.03 |
| maxIntegral | 1.02 | 1.03 | 1 | 1.02 |

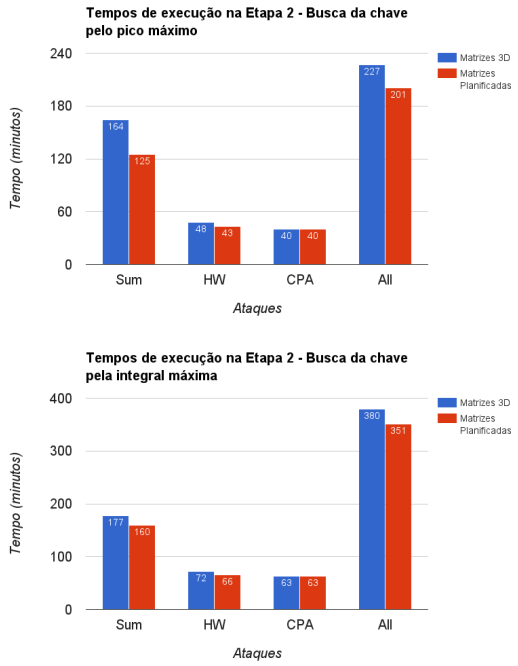


**Figura 7. Comparação dos tempos de execução entre o algoritmo com as matrizes tridimensionais e as matrizes planificadas na Etapa 2.**

### 5.4. Paralelismo com OpenMP

Na quarta e última etapa de otimização proposta é implementado o paralelismo utilizando OpenMP. Considerando que DPA possui matrizes globais que são incrementadas a cada vez que uma hipótese de chave é percorrida, não é possível aplicar um paralelismo em níveis mais externos do aninhamento dos laços `for` neste algoritmo. Desta forma a estratégia utilizada é paralelizar o laço das hipóteses de chave, considerando o grande número de dependência de dados existente no algoritmo.

As estruturas utilizadas para paralelizar o laço das hipóteses de chave são as diretivas `#pragma omp parallel for`, e também é especificado com a diretiva `firstprivate`, onde cada `thread` disparada pelo `parallel for` terá sua própria instância do vetor que possui o conteúdo do traço lido de entrada e seu próprio valor intermediário. Ao usar uma `thread` observa-se que em todos os ataques há um aumento em seu tempo de execução, isso demonstra que o custo da criação e sincronização das `threads` em OpenMP apresentam um custo significativo que não deve ser ignorado.

Diferentemente da etapa anterior, o paralelismo trouxe bons ganhos de desempenho na execução do algoritmo. Os tempos estão demonstrados na Figura 8 onde observa-se que o número de `threads` 0 representa o tempo de execução do algoritmo com otimizações na Etapa 3, os números de `threads` seguintes são os números de `threads` utilizados com o OpenMP.

A busca pela integral máxima possui o mesmo comportamento que a busca pelo pico máximo, porém somente em Sum e ALL houve perdas de desempenho ao utilizar as 4 threads. Novamente observou-se que CPA teve seu desempenho elevado e o HW também teve um pequeno ganho de desempenho ao utilizar 4 `threads`. Na Tabela 3 são

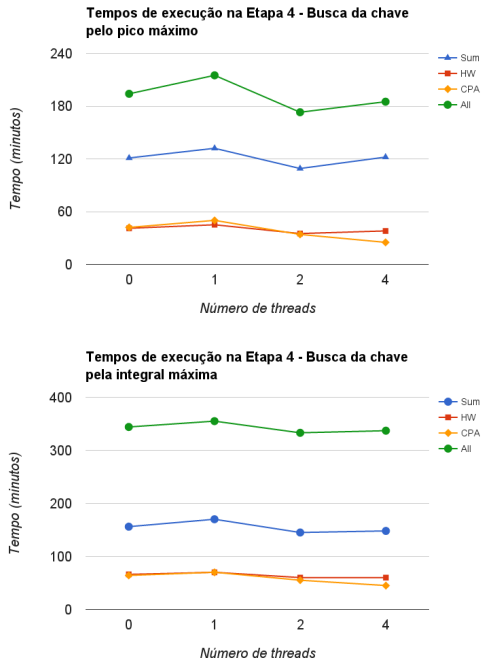


**Figura 8. Comparação dos tempos de execução do algoritmo com as diretivas SIMD e do algoritmo paralelo na Etapa 4.**

**Tabela 2. Speedups do algoritmo paralelo na Etapa 4 com busca pela chave máxima e integral máxima, respectivamente.**

| | Sum | HW | CPA | ALL | Sum | HW | CPA | ALL |
|---|---|---|---|---|---|---|---|---|
| 1 Thread | 0,9 | 0,9 | 0,8 | 0,9 | 0,9 | 0,9 | 0,9 | 0,9 |
| 2 Threads | 1,1 | 1,3 | 1,2 | 1,1 | 1,1 | 1,3 | 1,2 | 1,03 |
| 4 Threads | 0,99 | 1,1 | 1,4 | 1,02 | 1,05 | 1,1 | 1,4 | 1,02 |

representados os speedups calculados para cada `thread` nesta última etapa de otimização do algoritmo em relação aos resultados obtidos na Etapa 3.

## 6. Conclusão

Neste trabalho são propostas e avaliadas quatro propostas de otimização para uma implementação do algoritmo de Análise Diferencial de Potência (DPA) desenvolvido em [Lomné et al. 2009]. O trabalho inicialmente revisa as principais motivações para sua execução bem como algumas ferramentas para programação paralela que podem explorar o paralelismo de execução do algoritmo DPA. Além disso, a revisão mostra que por lidar com problemas de natureza aninhada, muitas diretrizes disponíveis ao programador e um alto nível de abstração levam a escolha pela interface OpenMP.

Os experimentos avaliam a execução do algoritmo para cada nível de otimização proposto. Os resultados demonstram significativos ganhos em desempenho, principalmente no ataque CPA, onde houve uma redução de aproximadamente 160 vezes em seu tempo de execução, usando a busca pelo pico máximo. Além disso, os experimentos destacam relevantes ganhos de desempenho na ordem de 5 vezes para ataques realizados com

SUM e 13 vezes para ataques usando o parâmetro HW. É preciso enfatizar, que a maior parte desse ganho, entre todas as fazes de otimização, provém da tradução do algoritmo de Matlab para C++.

**Tabela 3. Tempos de execução finais em minutos, com busca pelo pico máximo e integral máxima, respectivamente.**

| | Sum | HW | CPA | ALL | Sum | HW | CPA | ALL |
|---|---|---|---|---|---|---|---|---|
| Matlab | 570 | 450 | 4.020 | 10.200 | 540 | 420 | 3.950 | 10.320 |
| C++ | 109 | 35 | 25 | 173 | 145 | 60 | 45 | 333 |

Com as implementações otimizadas, é possível executar um ataque DPA em um número maior de traços, de forma a obter-se resultados em um intervalo menor de tempo. Isto permite explorar vários cenários de ataques em um tempo viável, avaliando ainda mais a fuga de informações por este canal lateral, reduzindo o tempo de espera pelos resultados, que pode ser na ordem de dias e até mesmo semanas para o fim de sua execução.

## Agradecimentos

O presente trabalho foi realizado com apoio do Programa Nacional de Cooperação Acadêmica da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES/Brasil.

## Referências

Bartkewitz, T. and Lemke-Rust, K. (2011). A High-Performance Implementation of Differential Power Analysis on Graphics Cards. In *CARDIS*, pages 252–265. Springer.

Brier, E., Clavier, C., and Olivier, F. (2004). Correlation Power Analysis with a Leakage Model. In *Cryptographic Hardware and Embedded Systems-CHES 2004*, pages 16–29. Springer.

Gamaarachchi, H., Ragel, R., and Jayasinghe, D. (2014). Accelerating Correlation Power Analysis using Graphics Processing Units (GPUs). In *Information and Automation for Sustainability (ICIAfS), 2014 7th International Conference on*, pages 1–6. IEEE.

Kocher, P., Jaffe, J., and Jun, B. (1999). Differential power analysis. In *Advances in Cryptology—CRYPTO'99*, pages 388–397. Springer.

Kocher, P. C. (1996). Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, other Systems. In *Advances in Cryptology—CRYPTO'96*, pages 104–113. Springer.

Lomné, V., Maurine, P., Torres, L., Robert, M., Soares, R., and Calazans, N. (2009). Evaluation on FPGA of Triple Rail Logic Robustness against DPA and DEMA. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 634–639. European Design and Automation Association.

Stallings, W. (2008). *Criptografia e Segurança de Redes*. Pearson Prentice Hall.

Swamy, T., Shah, N., Luo, P., Fei, Y., and Kaeli, D. (2014). Scalable and Efficient Implementation of Correlation Power Analysis using Graphics Processing Units (GPUs). In *Proceedings of the Third Workshop on Hardware and Architectural Support for Security and Privacy*, page 10. ACM.

Waddle, J. and Wagner, D. (2004). Towards Efficient Second-Order Power Analysis. In *Cryptographic Hardware and Embedded Systems-CHES 2004*, pages 1–15. Springer.