Instituto Superior Técnico
Mestrado Integrado em Engenharia
Electrotécnica e de Computadores

Co-Projecto Hardware/Software

Desencriptação AES - Parte 2

David Fialho n.º 73530
Ricardo Amendoeira n.º 73373

Lisboa, 7 de Junho de 2015

Conteúdo

1 Introdução

Na primeira parte do trabalho foi feito um sistema para desencriptar ficheiros encriptados com o protocolo *AES* (Advanced Encryption Standard). O sistema usava um processador *MicroBlaze* e um acelerador por hardware, para reduzir o tempo de execução.

Na segunda parte o objectivo é melhorar a performance recorrendo a múltiplos processadores (e múltiplos aceleradores) e à paralelização do algoritmo. Nas *FPGA's*, ao contrário dos microprocessadores *general purpose* actuais, o incentivo para a paralelização não é o menor consumo mas sim o aproveitamento da àrea dísponível na FPGA, assim como a grande dificuldade em aumentar a frequência de operação.

2 Arquitectura de Paralelização

2.1 Divisão de Trabalho

O algoritmo para desencriptar *AES* (pode ser consultado no relatório da parte 1) não tem dependências entre cada bloco de 16 *bytes*, pelo que a divisão de trabalho entre os vários processadores é muito simples: Cada processador fica encarregue de aproximadamente N/P blocos, em que N é o número de blocos e P o número de processadores usados. De forma mais formal, identificando cada bloco com um número inteiro $i \in [0, N[$ e cada processador com um número inteiro $p \in [0, P[$, o processador p fica encarregue de todos os blocos para os quais $i \mod P = p$.

Um dos processadores é considerado o *master* e está encarregue, para além de fazer a sua parte da computação, de transmitir os resultados para o utilizador, sinalizar os outros processadores para iniciarem a computação e de gerir o *timer* de execução, que é usado para medir a performance do sistema

2.2 Sincronização

Como já referido, não há dependências entre blocos de 16 *bytes* do ficheiro, pelo que a sincronização necessária é mínima: basta a existência de uma forma de o *master* confirmar que todos os processadores chegaram ao fim de execução, para que o sistema consiga saber quando a computação está completa. É também usada uma barreira antes de começar a computação mas apenas devido ao uso do *timer*, não é necessária para a correcta execução do algoritmo.

A sinalização ao *master* é feita da seguinte forma: cada processador tem uma entrada na memória partilhada que é inicializada com um determinado valor. Quando o processador termina a computação escreve um valor diferente nessa entrada de memória, sinalizando que terminou. O *master*, chegando ao fim da sua carga de trabalho, monitoriza estas posições de memória até confirmar que todos terminaram, altura em que o

sistema dá como completa a computação, termina a contagem de tempo e transmite o resultado para o utilizador.

3 Implementação

3.1 Características do Hardware

3.1.1 Dimensionamento

Nesta segunda implementação fez-se um esforço de forma a tentar reduzir o tamanho do programa com o objectivo poder colocar tanto o programa como os dados e o *stack* respectivos na memória local de cada processador. Se isto não fosse possível era necessário garantir que o espaço de programa, dados e *stack* de cada um dos processadores não se sobrepunham, desta forma este requisito é explicitamente garantido. Além de preencher este requisito a utilização das memórias locais permitiu acelerar a execução do programa em aproximadamente 10 vezes.

Foi feita uma estimativa do tamanho necessário para o *stack* tendo em conta as funções chamadas durante a execução da aplicação e concluiu-se que é necessário um *stack* com, aproximadamente, 2KB.

Tendo em conta o tamanho do *stack* e dado que o programa em cada um dos processadores tem um tamanho de, aproximadamente, 20KB foi necessária a utilização de memórias locais de 32 KB.

O *master* tem um tamanho de programa algo superior aos outros processadores (24KB), isto deve-se principalmente à utilização do *"xil_printf"*, mas além desta função este inclui algum código extra para assegurar a sincronização e a contagem do tempo. A função *"xil_printf"* é utilizada apenas como método para imprimir o tempo de execução e o resultado da computação, desta forma numa aplicação final este não é utilizado e por isso pode considerar-se que o tamanho de programa do *master* é essencialmente igual ao de todos os outros processadores.

A memória externa mantém-se ligada ao *microblaze* através de uma cache. Nesta segunda parte, dado que o programa se encontra armazenado em memória local a cache de instruções nunca é utilizada, sendo assim optou-se pela sua desactivação (na realidade não é possível desactivar, pelo que foi escolhido o tamanho mínimo). Dada a natureza da aplicação, verificou-se que uma vez que um bloco de 16 bytes era lido da memória, processado e nunca mais era acedido. Logo, a cache de dados de cada um dos processadores precisa apenas de ter capacidade para armazenar o conteúdo de um bloco inteiro de 16 bytes, após o processamento do bloco em cache este pode ser substituído pelo próximo bloco a processar uma vez que não volta a ser acedido. Definiu-se assim a cache de dados com a capacidade mínima de 64 bytes.

Além deste detalhe da aplicação, também se teve em conta que durante o processamento do bloco eram feitas várias escritas no espaço de memória correspondente a esse bloco. O ideal seria ter uma situação em que o bloco era carregado integralmente, processado (escrito) sempre em cache e quando finalmente o bloco se encontrasse no seu estado final este era escrito para a memória externa e o próximo bloco era carregado

para a cache no seu lugar. Para conseguir obter este funcionamento definiu-se o método *write-back* como método de escrita da cache de dados. Este método permite obter o funcionamento pretendido uma vez que os dados só são escritos na memória externa quando o bloco é substituído na cache. Este funcionamento permite reduzir o número de acessos à memória por parte de cada processador que é um dos factores mais limitativos da implementação.

Foram utilizados 4 processadores na implementação desta aplicação. O número de processadores utilizado foi escolhido com base nos resultados obtidos com um número variado de processadores e tendo em conta as limitações da FPGA disponibilizada no laboratório.

Para implementar a sincronização foi necessária a utilização de uma memória interna partilhada por todos o processadores através do barramento AXI-lite. Esta a memória é utilizada apenas para realizar a sincronização entre os vários processadores de acordo com o descrito na sub-secção de Sincronização da Arquitectura de Paralelização. Desta forma o acesso a esta memória é feito a partir do barramento AXI-lite para não interferir com o acesso ao barramento AXI-full utilizado para aceder à memória externa durante o processamento dos blocos.

3.1.2 Limitações

Uma das maiores limitações na utilização de múltiplos processadores deve-se ao acesso aos vários recursos partilhados. Na implementação realizada existem dois recursos partilhados: a memória interna e a memória externa. O recurso cujo o acesso é mais frequente e que apresenta uma maior limitação no desempenho da aplicação é a memória externa. Esta é acedida por todos os processadores sempre que cada um deste termina o processamento de um bloco e inicia o processamento do bloco seguinte. Este acesso é muito frequente uma vez que o processamento de cada bloco a relativamente rápido quando comparado com o tempo de acesso à memória externa para obter o bloco a processar.

Temos assim um acesso de N processadores para 1 recurso (memória externa). Tendo em conta que o barramento AXI permite que seja lido um bloco de 4 bytes em cada acesso à memória, para obter um bloco de 16 bytes cada processador tem que fazer 4 acessos à memória externa. Vê-se assim que o número de acessos à memória pode ser elevado e cresce linearmente com o número de blocos de 16 bytes a processar.

Com a utilização de apenas um processador não é necessário recurso a um árbitro e por isso o acesso à memória é directo e não exige lógica adicional. Com a utilização de dois processadores a concorrência no acesso à memória exige a utilização de um árbitro para discernir entre os dois processadores qual é que deve aceder ao recurso em cada instante. Um componente adicional entre o processador e a memória introduz *delay* de arbitragem adicional que cresce com a introdução de um maior número de processadores. Assim espera-se que o *speedup* esteja de acordo com o número de processadores até um certo ponto, a partir desse ponto o *speedup* começará a descer até que para um número muito elevado de processadores o *speedup* poderá ser menor do que 1.

3.2 Características do Software

O software foi implementado de forma a ser escalável para um qualquer número processadores possíveis. Assim sendo, para que o código utilizado com 4 processadores funcione para um número X de processadores basta indicar o número X de processadores que se pretende utilizar. No nosso caso isto é feito através de um define e por isso o programa tem que ser recompilado. Em certas aplicações o custo de recompilação pode ser elevado, por isso nesses casos o ideal seria ter uma forma de indicar o número de processadores em tempo de execução. Isto seria relativamente simples de implementar introduzindo alguma mecanismo de input no sistema. No entanto pensa-se que esta não seja uma grande preocupação no desenvolvimento deste tipo de sistemas uma vez que esta recompilação só tem que ser feita quando se pretende fazer alterações ao hardware e estas têm um custo temporal substancialmente mais elevado e não são comuns.

4 Resultados

Com a introdução de N processadores, teoricamente, espera-se obter um *speedup* de N em relação à implementação com apenas 1 processador. Esta expectativa assume que a introdução de mais processadores resulta numa execução completamente em paralelo, que não existem *overheads* de paralelização e que não existem limitações no acesso a recursos partilhados pelos processadores.

No desenvolvimento desta aplicação tentou-se a utilização de 2, 4, 8 e 12 processadores. Os resultados obtidos encontram-se nas tabelas abaixo.

# Processadores	448 bytes	52741 bytes	1235204 bytes	Speed Up	Eficiência
1	15	1765	41357	1	1
2	8	891	20703	1.95	0.98
4	4	448	10361	3.89	0.97
8	3	253	5896	6.33	0.79
12		Não testado			

Tabela 1: Resultados

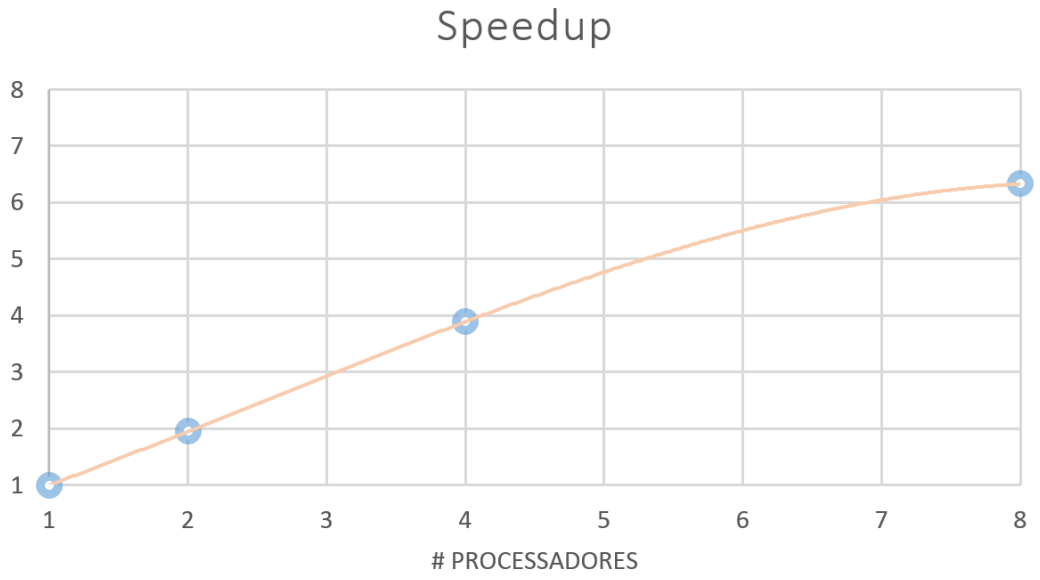

Figura 1: Speed Up

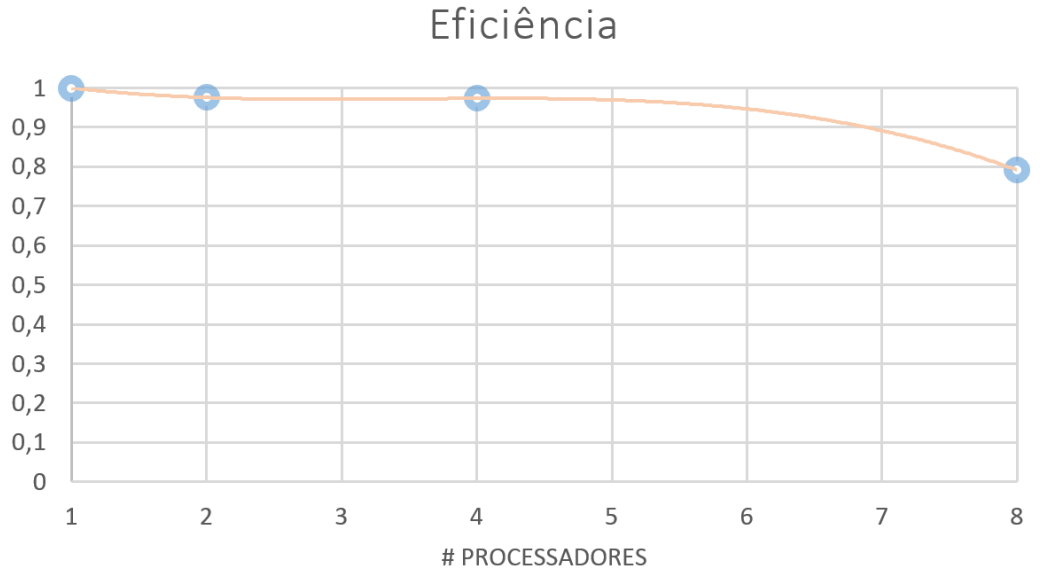

Figura 2: Eficiência

Numa fase inicial optou-se por fazer uma implementação apenas com 2 processadores para garantir que as técnicas utilizadas para implementar a distribuição de dados e a sincronização entre processadores funcionavam correctamente. Tal como se esperava, obteve-se um *speedup* de 2 em relação à implementação só com um processador, para ambos os testes com o ficheiro pequeno e com o ficheiro grande.

De seguida experimentou-se utilizar 4 processadores e os resultados obtidos continuaram de acordo com o esperado uma vez que o *speedup* obtido para ambos os testes foi aproximadamente 4.

Com a introdução de 8 processadores os resultados obtidos já não foram de acordo com o esperado, pois os *speedups* obtidos são consideravelmente mais reduzidos do que o esperado.

Antes de se tentar gerar o *bitstream* do hardware com 12 processadores foram feitas algumas contas para garantir que o número de BRAMs disponíveis era suficiente para as necessidades de memória da aplicação e chegou-se à conclusão que estas eram garantidas. Quando se tentou gerar o *bitstream* do hardware verificou-se então que o número de LUTs da FPGA era demasiado reduzido para gerar o *hardware* e por isso não foi possível testar o funcionamento com 12 processadores.

Com a posterior análise do relatório de síntese da implementação com 8 processadores concluiu-se que esta ocupava 99% do número de *slices* da FPGA, desta forma concluiu-se que este seria o limite máximo de processadores que poderiam ser utilizados na FPGA utilizada no laboratório.

A partir dos resultados obtidos concluiu-se que a teoria definida inicialmente não é verificada na prática para qualquer número de processadores. Até 4 processadores os resultados obtidos foram muito próximos do esperado, mas para quantidades superiores os resultados começam a afastar-se do ideal. Isto deve-se aos *overheads* de paralelização, que incluem principalmente o sincronismo e as limitações no acesso a recursos partilhados, nomeadamente a memória externa que é acedida constantemente por todos os processadores.

No caso da aplicação implementada os *overheads* de paralelização são muito reduzidos e por isso a maior limitação de escalabilidade encontra-se na arbitragem do acesso à memória de dados. Estes resultados estão de acordo com as limitações referidas na secção de Limitações do Hardware.

5 Conclusão

Ao passar a usar as instruções para a memória interna a *performance* melhorou em uma ordem de magnitude. Tal só poderia ser atingido através de paralelismo se fossem usados pelo menos 10 cores, pelo que a memória usada para diferentes tarefas no sistema é dos factores mais importantes quando se quer melhorar o tempo de execução.

No entanto, depois de optimizar este e outros factores no sistema *single-core* (por exemplo a criação de aceleradores por *hardware*, como na parte 1), pode ainda haver muito desempenho a atingir com a utilização de múltiplos *cores*, sendo este o foco da parte 2 do trabalho.

No caso do algoritmo de desencriptação *AES* a distribuição de tarefas e sincronização é muito simples, pelo que a maior parte do trabalho é no design do *hardware*. De notar também que o sistema tem complexidade $O(n)$ para qualquer número de processadores.

Como o algoritmo é altamente paralelizável, quase todas as limitações ao *speedup* são de *hardware* e, consequentemente, difíceis de resolver sem mudar de sistema. Como

mencionado nas secções anteriores, as maiores complicações no sistema usado são a àrea disponível na FPGA (especificamente a quantidade de LUT's) e o overhead no acesso ao bus AXI-FULL da memória externa quando é usado um número relativamente elevado de processadores.

Uma melhor utilização das LUT's, embora possível, exige muito tempo de design e tem retornos muito reduzidos (talvez seja possível incluir mais um ou dois processadores), pelo que a atenção deve estar na gestão do bus AXI-FULL. No caso do algoritmo usado seria possível dividir os dados de entrada pelos vários P processadores, tendo cada um a sua própria memória de dados (com uma dimensão P vezes menor que a memória externa usada neste trabalho). Desta forma o overhead de arbitragem do acesso à memória (o principal limitador de performance neste trabalho) iria ser praticamente nulo, permitindo que o *speedup* continuasse a ser proporcional ao número de processadores. Este método tem no entanto a desvantagem de tornar mais complexa a apresentação dos resultados mas não é um problema difícil de resolver.

A implementação de multi-processadores em FPGA's é portanto bastante útil, uma vez que é uma forma de melhorar *performance* relativamente simples (dependendo do algoritmo). Sendo possível combinar paralelismo com outras soluções, como a utilização de *hardware* dedicado, as desvantagens são reduzidas e é portanto uma solução fácil de recomendar.