Arquiteturas Reconfiguráveis e GARP

Karina Z. de Oliveira (028520)
Instituto de Computação, UNICAMP
Caixa Postal 6176
13084-971 Campinas, SP - BRASIL
+55 019 32557549
karinazupo@yahoo.com.br

RESUMO

O propósito deste trabalho é apresentar uma visão geral sobre o GARP. O GARP é um processador híbrido resultado de estudos sobre arquiteturas reconfiguráveis. Arquiteturas reconfiguráveis é um tópico que vem sendo muito estudado nos últimos anos, e que busca soluções para melhoria de desempenho nos computadores, através de unidades lógicas reprogramáveis (FPGAs) conectadas a processador comum. Um dos principais objetivos do projeto GARP é elaborar e obter conclusões sobre um processador com arquitetura reconfigurável que possa ser incorporado aos computadores *desktop* atuais e ser usado em aplicativos genéricos. A seguir breve descrição sobre a arquitetura interna do GARP, o suporte a software e algumas conclusões.

Palavras Chave

Arquiteturas reconfiguráveis, FPGA, GARP, *desktop*.

1. INTRODUÇÃO

Os computadores de propósito gerais executam instruções de forma extremamente otimizada. Primeiramente veio o conceito de *pipeline*. Depois vieram os processadores superescalares e VLIW. Esses computadores são bons para executar instruções em geral, de modo a ter um bom desempenho em *workloads*. Mas, como otimizar ainda mais? Executando código em unidades de lógica reprogramáveis.

1.1 ASICs

Os ASICs são circuitos integráveis lógicos específicos para uma determinada aplicação, ou seja, executam um determinado algoritmo. Justamente o contrário dos processadores programáveis, que executam um conjunto de instruções em qualquer sequência. Assim, suponhamos que se quer executar o cálculo de "(a + b) * (c + d)". Um ASIC para realizar tal cálculo seria algo como o que está esquematizado na Figura 1. Como não é preciso decodificar as instruções, buscá-las em memória, e atravessar todo o *datapath*, os ASICs executam de forma mais rápida para os problemas específicos que se propõem a solucionar. Os ASICs são elaborados e configurados durante sua fabricação e não podem ser alterados depois, não podendo portanto ser usados para aplicações de propósito geral como as utilizadas em computadores *desktop*. Entretanto, existem os FPGA que podem implementar circuitos no estilo ASIC, enquanto por outro lado, podem ser reprogramados sempre que desejar-se, podendo ser usados assim para aplicações de propósito geral. Como seria possível usar as vantagens do conceito de ASIC através de FPGAs em aplicações de propósito geral?

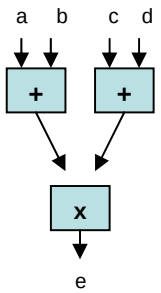

Figura 1: Circuito ASIC esquematizado

1.2 FPGAs

FPGA é sigla de *Field Programmable Gate Array*. Um FPGA é um circuito integrado que pode ser programado e reprogramado depois de fabricado, em geral por meio de software. Os dispositivos FPGAs consistem de um *array* de blocos lógicos configuráveis, que implementam funções lógicas AND, NAND, OR, NOR e XOR. Os blocos lógicos possuem estruturas similares aos *gate arrays* usados em alguns circuitos ASIC. Veja um esquema na Figura 2.

Os primeiros FPGAs ocupavam um "grande espaço" e necessitavam de vários segundos ou mais para alterar seus conteúdos, o que é razoável para verificar novos protótipos em um ciclo de desenvolvimento. Porém, seria inviável usá-los para em computadores *desktop*. Recentemente, tem se desenvolvido novos FPGAs cada vez menores e que podem ser configurados com velocidades extremamente altas. Assim, estudos recentes estão tentando integrá-los para uso em computadores *desktop*.

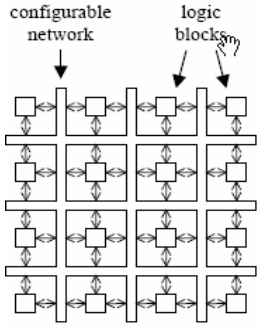

Figura 2: Estrutura de um FPGA

1.3 Arquiteturas Reconfiguráveis

Na década de 90, muitos projetos de arquiteturas reconfiguráveis foram propostos para otimizar aplicações específicas. Muitos estudos foram realizados utilizando placas com FPGAs, geralmente conectadas via PCI, para resolver problemas específicos como sequenciamento de genoma, criptografia e tratamento de imagens. Para implementar tais algoritmos, projetos foram elaborados com placas que possuiam vários FPGAs conectados, pois somente com um FPGA não era possível resolver o problema. Nesse tempo, os FPGAs ocupavam um espaço considerável e não podiam ser embutidos no *chip* do processador. Além disso, esses FPGA mais antigos possuiam um tempo de configuração grande, e assim, para aplicações específicas, em que a máquina era configurada somente na inicialização, esses tempos eram aceitáveis. Porém, esses tempos eram demasiado grandes para que os FPGA fossem

continuamente reconfigurados para cada processo genérico executado por máquinas de propósito geral. E por estarem em placas separadas, além dos tempos de reconfiguração, ainda existem os tempos para comunicação, feita através de barramentos, com o processador.

1.4 Projeto Garp

Atualmente, os FPGAs são pequenos o suficiente e podem ser incluídos no *chip* do processador. Pequenos o suficiente a ponto de existirem pesquisas onde os FPGA são adicionados como unidades funcionais dentro do *datapath* do processador. Também, o tempo de configuração é menor, e, com isso, esse tempo de configuração poder ser amortizado pelo grande número de usos repetidos de uma determinada programação.
O projeto GARP[1][2] estuda como combinar um processador MPIS-II com um co-processador FPGA num mesmo *chip*. O projeto Garp também é adepto do seguinte corolário: "90% do código de um programa executa por somente 10% do tempo de execução, sendo assim, 10% do código de um programa é executado por volta de 90% do tempo!" Esses trechos de código que executam por muito tempo são chamados de *kernéis* do programa. A proposta é então executar esses trechos de código em unidades de lógica reconfigurável e estudar os ganhos de desempenho obtido com isso.

A pesquisa do projeto GARP foca em:

- aplicar o uso do co-processador FPGA em pequenos trechos de código que gastam a maior parte do tempo de execução de um programa, provavelmente os laços, e continuar usando o processador genérico para os outros trechos de código;
- o tempo de reconfiguração deve ser minimizado, e o co-processador FPGA deve ter as mesmas facilidades de acesso à *cache* e à memória do próprio processador.
- o FPGA deve estar contido no mesmo *chip* do processador, para que possa ser vendável no mercado de *desktops*.

E mais: como o Garp se comporta em ambientes multitarefa, onde processos são interrompidos e outros processos são reiniciados em qualquer momento? Como o co-processador FPGA lida com acessos a memória virtual e eventual falhas de página? Além disso, é preciso prover compatibilidade binária entre os diversos FPGAs.

2. ARQUITETURA

O GARP é composto de um processador MIPS-II com uma unidade de FPGA ligada a este processador, tudo num mesmo *chip*. A *thread* principal de um programa é executada pelo processador MIPS-II e é o próprio programa quem controla quando uma programação de trecho de código será carregada e executada no FPGA. Uma programação a ser executada no FPGA também é chamada de configuração. Na arquitetura do Garp, FPGA tem acesso direto ao *cache* de dados e à memória. A Figura 3 mostra um esquema desse *chip*.

O FPGA é composto por blocos. Esses blocos são organizados em linhas de 23 blocos lógicos e um bloco de controle. O número de linhas de uma configuração pode ser variável. Os FPGAs utilizados são os Xilinx da série 4000 (www.xilinx.com). Os blocos de controle são blocos por onde cada uma das linhas do FPGA se comunica com a memória ou com o processador. Blocos de controle podem iniciar acessos à memória e podem parar a execução do FPGA.

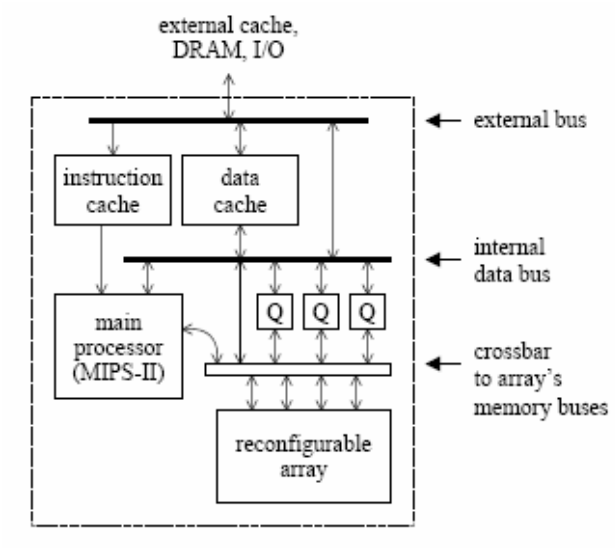

Figura 3: Diagrama de bloco do GARP

Quatro barramentos de memória atravessam verticalmente as linhas de blocos do FGPA e são usados para configuração e para a transferência de dados entre a memória e o FPGA diretamente. O FPGA é interrompido (*stalling*) de forma transparente nos acessos a memória.

Fios *(wires)* verticais e horizontais são usados para fazer a conexão entre os blocos. Não há conexões entre fios que não passem pelos blocos lógicos. Um esquema do FPGA é mostrado na Figura 4. Curiosidade: os fios ocupam o espaço mais significativo dentro de um FPGA.

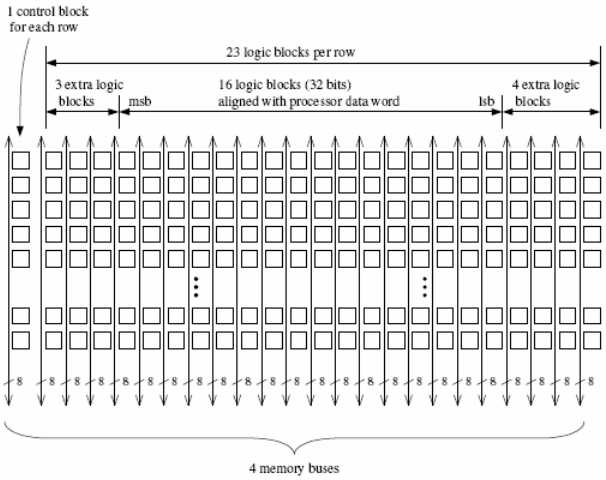

Figura 4: Esquema deo FPGA

Para o carregamento de configurações e para a transferência de dados entre o processador e o FPGA foram adicionadas instruções ao MIPS. Essas instruções são: *gaconf*, para carregar uma configuração no FPGA, *mtga* que copia o conteúdo de um registrador do processador para um registrador do FPGA e *mfta*

que copia o conteúdo de um registrador do FPGA para um registrador do processador.

Programas genéricos são grandes, e o FPGA possui recursos de espaço limitados, assim, as configurações devem ser codificadas tão densas quanto o possível. Com isso os tempos de carregamento da configuração são menores e *cache* de configuração é mais bem aproveitado. Configurações com erros podem danificar o FPGA e uma verificação da configuração do FPGA é feita no momento do carregamento, pois configurações não seguras podem causar danos físicos no FPGA. Somente uma configuração está ativa num determinado momento, e uma vez carregada, a configuração não pode ser alterada.

Existe um *cache* com as configurações do FPGA recentemente usadas. Isso possibilita trocas de contexto entre processos mais rápidas, requisito indispensável para ambientes multitarefa. Essa *cache* funciona de modo transparente aos programas, e a idéia é similar ao *cache* de instruções. Esse *cache* suporta o equivalente a 128 linhas de blocos, distribuídos em 4 últimas configurações usadas para cada linha física de blocos. Para a troca de contexto, foram adicionadas as instruções *gasave* para salvar o estado interno do FPGA para a memória e *garestore* para restaurar o estado interno do FPGA da memória.

Cada bloco lógico do FPGA implementa uma função com 4 entradas de 2 bits. Um bloco recebe 4 entradas, chamadas A, B, C e D, e calcula duas saídas de 2 bits, Z e D (que também pode ser a saída direta da entrada D). Cada bloco possui dois registradores, onde podem ser armazenados as saídas ou valores vindos da memória. E valores armazenados nestes registradores podem ser despachados para a memória. O FPGA deve possuir suporte a aritmética de bit paralela e a propagação de *carry* rápida. Com isso, adições, subtrações e multiplicações podem ser executadas de maneira mais eficiente. A Figura 5 mostra o esquema de um bloco do FPGA.

Existem possíveis lógicas de bloco que podem ser utilizadas nas configurações: *table-lookup*, *triple-add*, *select*, *split table*, *carry chain* e *partial select*. Esses tipos de lógicas de blocos serão usados na programação da configuração do FPGA.

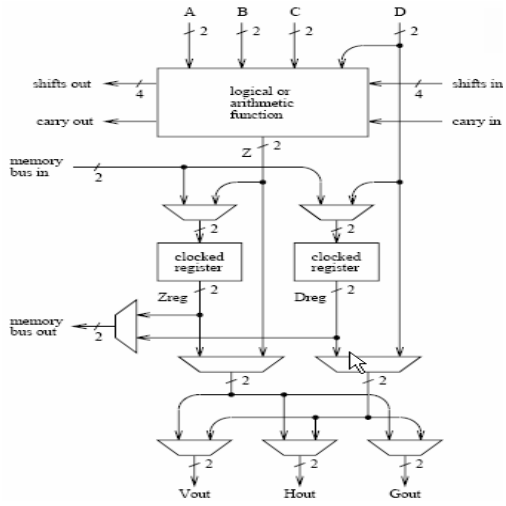

Figura 5: Esquema de um bloco do FPGA

Acopladas ao FPGA existem 3 filas de memória de onde o FPGA pode escrever e ler diretamente, e são acessíveis de forma simultânea. Da perspectiva do FPGA, acessos às filas são como acessos à memória, sem a necessidade de prover endereços. O comportamento e conteúdo das pilhas deve ser programando antes da configuração ser executada. As instruções *galqc* e *gasqc* podem ser usadas pelo processador para manipular essas filas.

O ciclo de do FPGA é definido em termos de que operações podem ser executadas dentro de um ciclo. Quantificar o tempo com essa regra facilita o cálculo do número de ciclos de *clock* que um determinado processamento precisará. Isso permite que a mesma configuração possa ser executada sem alterações em diferentes implementações de FPGAs provendo assim compatibilidade binária.

Um contador de *clock* controla a execução do FPGA. Enquanto o contador é diferente de zero, ele é decrementado em um a cada ciclo de *clock*. Quando o contador chega a é zero, alterações no FPGA são interrompidas, e o FPGA é parado. O processador altera o contador de *clock* do FPGA para um número específico de passos. O próprio FPGA pode zerar o contador para indicar que a computação que estava sendo nele efetuada terminou. O processador pode manipular o contador de *clock* do FPGA com as instruções *gabump* e *gastop*.

3. SOFTWARE

Para desenvolver programas para Garp, ou seja, para desenvolver programas que fazem uso do FPGA especialmente projetado para o Garp, existe uma ferramenta chamada *configurator*. O *configurator* recebe como entrada uma linguagem, em formato textual, onde são definidas todas as linhas de uma configuração de FPGA, descrevendo os blocos lógicos de cada coluna. A sintaxe básica é:

```
row optional-row-name:
{
  column-number(s):logic-block-settings;
}
```

O *configurator* gera como saída um vetor de inteiros de 32 bits, como por exemplo:

```
{
 0x00000002,0x00000000,0x00000008,0x00000000,
 0x00000000,0x00000000,0x00000000,0x00000000,
 0x00000000,0x0A00000E,...
}
```

O FPGA deve ser utilizado em blocos de código do programa que gastam muito tempo de processamento, e que o ganho desempenho seja suficiente para que o tempo de carregamento da configuração do FPGA seja justificado. O resto do programa continua sendo desenvolvido em linguagem C, e compilado como um programa comum em C, que é executado no processador comum.

Continuando, o próximo passo seria utilizar a saída do *configurator* para iniciar um vetor de inteiros no programa escrito em C:

```
uint32 config_add[] = #include "add.config";
```

Isso torna acessível ao programa C a configuração do FPGA. O programa é executado normalmente no processador comum, e

quando chega o momento, é o próprio programa que carrega a configuração no FPGA e inicia sua execução. O carregamento da configuração no FPGA, execução, e a obtenção dos resultados são feitos via código *assembler*, incluído no próprio código do programa em C. É claro que é necessário que o montador (*assembler*) seja alterado para traduzir as instruções específicas do Garp para linguagem de máquina.

Abaixo segue um exemplo de código *assembler* que deve ser incluído no programa C para usar uma determinada configuração de FPGA:

```
# Load v0 with pointer to config_add3 array.
add3: la v0,config_add3
# Load configuration.
gaconf v0
# Copy three operands to array...
mtga a0,$z0
mtga a1,$d0
# And step array 2 clock cycles.
mtga a2,$d1,2
# Copy result back from array.
mfga v0,$z1
# Return from subroutine.
j ra
```

Um processador Garp não existe concretamente. Um programa Garp é executado e testado através de um simulador. Esse simulador executa programas MIPS. O *chip* do Garp se baseia, hipoteticamente, num processador Sun UltraSPARC, onde as unidades de processamento de inteiro e ponto flutuante foram substituídas por um processador MIPS-II de 32 bits conectado com um FPGA. As *caches* de instruções e de dados permanecem as mesmas do UltraSPARC. Ciclos, acessos à memória, acesso às filas de memória do Garp, percentual de falhas de *cache*, entre outros itens, são modelados.

Nos experimentos ficou claro que para aplicações altamente paralelizáveis, como tratamento de imagem, criptografia, sequenciamento de genomas, o uso do FPGA traz grande ganho de desempenho. Para outros tipos de aplicações, os ganho são pequenos, da ordem de 4 vezes[2].

3.1 Garpcc

Callahan[3] implementou um compilador C chamado `garpcc` adaptado para a arquitetura do Garp. Esse compilador recebe como entrada ANSI C, e o programador não precisa incluir quaisquer diretivas ou comandos para gerar código específico para o Garp. Este compilador procurar pelos por blocos de códigos dos laços candidatos a serem executados no FPGA, enquanto o resto do programa seria executado no processador comum.

Técnicas de compilação para VLIW(very long instruction words), muito estudadas, foram adaptadas para serem usadas no `garpcc`. Hiperblocos é um conceito utilizado pelo `garpcc` oriundo dessas técnicas.

O compilador SUIF foi utilizado para a compilação e otimizações padrões de código C, e para a obtenção de blocos básicos de código. SUIF[5] é um compilador baseado em técnicas para encontrar blocos de código independentes que possam ser executados paralelamente e em técnicas para manipular a memória compartilhada de modo eficiente por vários processadores.

Uma vez encontrados todos os blocos de código, como selecioná-los para executar no FPGA? Além de excluir blocos com chamadas a bibliotecas C (por exemplo, chamadas `printf`), ferramentas de *profiling* foram utilizadas para selecionar os melhores blocos a serem executados no FPGA. Além disso, o compilador também usa técnicas de *loads* especulativos, *pipelining* de execução de diferentes iterações de laços, e uso extensivo das filas de memória do Garp para simular acessos rápidos a memória.

Nos experimentos, utilizando o simulador do Garp, Callahan[4] encontrou resultados não muito expressivos, devido a problemas de falhas de *cachê* de configuração, programas com muitos laços pequenos, saídas de laços e rotinas da biblioteca C não compiladas especialmente para o uso no Garp. Os melhores resultados foram programas que possuem grandes blocos de código paralelizáveis em muitos laços.

4. CONCLUSÕES

O projeto Garp contribuiu com investigações bastante valiosas sobre como integrar um FPGA num processador de um computador *desktop*. Foi elaborada uma arquitetura que consiste no uso de um FPGA no próprio *chip* do processador, com o mesmo acesso à hierarquia de memória que o processador comum. O FPGA possui filas de memória e um *cache* de configurações. O processador MIPS foi usado no estudo, e instruções especiais para uso do FPGA foram adicionadas. Também foram levados em conta problemas como trocas de contexto de processador, falhas de acesso à memória e compatibilidade binária.

Configurações para o FPGA podem ser desenvolvidas manualmente, através do *configurator*. Através de experimentos realizados com diversas aplicações pode-se avaliar qual foi o uso quantitativamente quais blocos lógicos foram mais utilizados, o uso das filas, o uso dos *wires* e os fatores limitantes. E ficou claro que para aplicações paralelizáveis o uso do FPGA traz ganhos de desempenho.

Para que programas reais possam ser desenvolvidos, muito ainda precisa ser feito. Callahan iniciou este trabalho desenvolvendo o compilador `garpcc`, porém, este compilador e a própria arquitetura do Garp precisam ainda de muitas melhorias para que o `garpcc` possa realizar compilações que obtenham bons ganhos de desempenho em aplicações genéricas.

Assim, para que processadores híbridos como o Garp sejam parte de computadores *desktop* e se tornem uma realidade, assim como os processadores superescalares já o são hoje, muitos itens ainda precisam ser estudados e respondidos.

5. REFERÊNCIAS

[1] Hauser, J. R., and Wawrzynek, J. Garp: *A MIPS Processor with a Reconfigurable Coprocessor*. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '97, April 16-18, 1997), pp. 24-33 (10 pages).

[2] Hauser, J. R *Augmenting a Microprocessor with Recongurable Hardware*. Ph.D. Thesis, University of California, Berkeley, 2000.

[3] Callahan, T. J., Hauser, J. R., and Wawrzynek, J. *The Garp Architecture and C Compiler*. IEEE Computer, April 2000.

[4] Callahan, T. J. *Automatic Compilation of C for Hybrid Reconfigurable Architectures.* Ph.D. Thesis, University of California, Berkeley, 2002.

[5] Hall , M. W., Anderson , J. M., Amarasinghe, S. P., Murphy, B. R., Liao, S.-W., Bugnion, E., and Lam, M. S. *Maximizing Multiprocessor Performance with the SUIF Compiler*. IEEE Computer, December 1996.