AVALIAÇÃO DE DESEMPENHO EM UMA NOC DE UMA APLICAÇÃO DISTRIBUÍDA UTILIZANDO UM MIDDLEWARE JAVA

Gustavo Siebra Lopes[1], Elias Teodoro da Silva Jr[2]

[1]Graduando em Engenharia de Telecomunicações - IFCE. Bolsistas do CNPq. e-mail: gustavosiebra@gmail.com
[2]Doutor em Ciências da Computação – UFRS. e-mail: elias@ifce.edu.br

Resumo: Esse trabalho consiste em um software de aplicação para uma plataforma multiprocessada voltada para sistemas embarcados. Para isto, foi usado um modelo de interface de programação que atua sobre uma plataforma MPSoC, voltada para sistemas embarcados e de tempo-real, um middleware chamado de MiddleSoC. O software implementado é um multiplicador de matrizes que pode servir de exemplo para outras aplicações distribuídas, escritas sobre o *middleware*. Esse trabalho analisa os resultados encontrados durante os experimentos e tira conclusões sobre o desempenho de um sistema multiprocessado, considerando o tempo gasto com troca de mensagens entre os processadores.
Palavras–chave: Java, MiddleSoC, MPSoc, NoC, RTSJ, Sistemas Embarcados

1. INTRODUÇÃO

Atualmente a complexidade dos sistemas embarcados tem se tornado cada vez maior, e para esses sistemas complexos, o ideal é que se use uma plataforma multiprocessada (MPSoC-Multiprocessor SoC). Com várias unidades de processamento trabalhando de forma distribuída, o sistema ganha desempenho sem elevar consideravelmente a potência dissipada. Entretanto, a distribuição traz consigo o problema de concorrência (GILL, 2003).

Esse trabalho foi desenvolvido em uma plataforma MPSoC de processadores FemtoJava, conectados pela NoC (*Network-on-Chip*) SoCIN (*System-on-Chip Interconnection Network*). Os códigos Java foram compilados por um compilador padrão Java e processados pelo ambiente de geração de aplicações específicas baseadas em software e hardware para microcontroladores, o SASHIMI (*System As Software and Hardware In MIcrocontrollers*). Para verificação do seu funcionamento foi utilizado o simulador Simple, com a opção de implementação em um FPGA (*Field Programmable Gate Array*) por síntese lógica. A implementação, um Multiplicador de Matrizes, levou em consideração que todo sistema multimídia utiliza matrizes como base fundamental em seu processamento, sendo este domínio de aplicação um forte candidato ao uso de MPSoC.

2. A PLATAFORMA

2.1 Hardware

O FemtoJava, inicialmente apresentado por Ito (2001), sofreu várias modificações ao longo do seu desenvolvimento em otimização, tamanho e na facilidade de programar. Atualmente, o processador é composto por uma unidade de processamento baseada em arquitetura de pilha, memórias RAM e ROM integradas, portas de entrada e saída mapeadas em memória e um mecanismo de tratamento de interrupções com dois níveis de prioridade. A máquina de pilha implementa um subconjunto de instruções da JVM (*Java Virtual Machine*), fazendo do FemtoJava uma JVM em hardware. As instruções são divididas em: operações com inteiros e de manipulação de bits, operações de leitura e armazenamento na memória, desvios condicionais e incondicionais, operações de pilha e dois pseudo-bytecodes para leitura e escrita arbitrária em memória. Seu funcionamento é consistente com a especificação da JVM.

Para usar esta plataforma, as classes Java da aplicação devem ser compiladas em uma ferramenta de compilação Java padrão e depois processadas pelo SASHIMI. O SASHIMI é o ambiente utilizado para a síntese de sistemas microcontrolados especificados em linguagem Java. As ferramentas do ambiente SASHIMI conseguem carregar arquivos executáveis Java (.class), interpretar

ISBN 978-85-62830-10-5

suas estruturas internas e, portanto extrair o código e outras informações necessárias à execução da aplicação. A ferramenta lê todas as classes da aplicação e gera um conjunto de arquivos VHDL (*Very High Speed Integrated Circuit (VHSIC) Hardware Description Language*), cujo conjunto de instruções contém somente aquelas utilizadas pela aplicação, que descreve o processador e as memórias ROM (métodos) e RAM (atributos). Esta descrição pode ser usada tanto para síntese lógica quanto para a simulação. O SASHIMI ainda pode ser considerado uma ferramenta de otimização da JVM, pois retira do código final os métodos e atributos que não são usados em momento nenhum pela aplicação.

2.1.1 NoC SoCIN

Com o aumento da complexidade de algumas aplicações de sistemas embarcados, a procura por sistemas de baixo consumo de energia, alto desempenho, flexibilidade e paralelismo simultaneamente também tem aumentado. O uso de sistemas multiprocessados (MPSoCs) é uma forte tendência para sistemas embarcados por oferecer elevado desempenho a um baixo custo em energia e potência.

MPSoCs (*Multiprocessor Systems-on-Chip*) são constituídos por vários processadores conectados por barramento ou por uma rede interna (NoC). No entanto, barramentos oferecem suporte limitado no que diz respeito à escalabilidade, reusabilidade, paralelismo e consumo de energia. Para resolver este problema foram propostas as NoCs por Benini (2002). As NoCs têm como principais vantagens: largura de banda escalável, uso de conexões ponto-a-ponto curtas e o paralelismo na comunicação. Embora tenham como desvantagens maiores custos e latência na comunicação, esses problemas vêm sendo reduzidos pela grande disponibilidade de transistores e por soluções arquiteturais que permitem diminuir a latência da rede e seus efeitos no desempenho da aplicação (ZEFERINO, 2003).

2.2 Software

2.2.1 Middleware

O middleware MiddleSoC foi proposto com o objetivo de prover o reuso da infraestrutura de hardware e software já existentes, aliando mecanismos que auxiliem na expressão de propriedades de tempo-real. Ele pode gerenciar adaptações em tempo-real, deixando o programador das aplicações livre dessa tarefa. Além disso, a complexidade das ações de comunicação entre processadores e entre hardware e software está encapsulada em primitivas de alto nível (SILVA JR., 2008a). A arquitetura do MiddleSoC está organizada em camadas com níveis de abstração diferenciados, buscando melhores resultados em parâmetros como energia ou memória, além de atender a restrições temporais. Esta arquitetura foi dividida em dois níveis organizacionais: nível de estrutura e nível de serviço (ver a Figura 1).

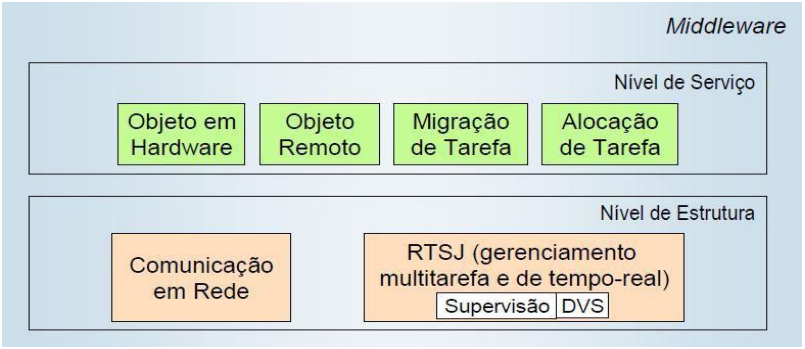


Figura 1 - Arquitetura do Middleware (SILVA JR., 2008a)

ISBN 978-85-62830-10-5

O nível de estrutura oferece os recursos mais básicos do *middleware*, que são a comunicação em rede e o gerenciamento multitarefa e de tempo-real. O nível de serviço oferece uma maior abstração e utiliza os recursos implementados no nível de estrutura, permitindo a exploração de diferentes arranjos para alocação das tarefas tanto em tempo de projeto quanto em tempo de execução nos projetos de aplicações embarcadas multiprocessador. Este trabalho explora mais os recursos do nível de estrutura do *middleware*. As seções seguintes vão detalhar o funcionamento dos serviços utilizados.

2.2.2 API de Tempo-Real

A RTSJ (*Real-Time Specification for Java*) (BOLLELLA, 2000) define um conjunto de especificações de comportamento em Java, que permite desenvolver aplicações de tempo real. A especificação permite ainda descrever: tarefas periódicas e esporádicas, orçamento de tempo de processamento e deadline de tarefas.

Na linguagem Java padrão, a aplicação é executada sobre o sistema operacional e quem faz o escalonamento das threads é a JVM. Na plataforma deste trabalho, quem faz as funções de gerenciamento de threads, típicas do RTOS (*Real-Time Operating System*), é a API de Wehrmeister (2004) e esta API é executada sobre a JVM, desempenhada pelo processador.

Como em qualquer sistema operacional de tempo real (RTOS), o escalonamento de tarefas consiste em um processo adicional no qual o processador é alocado a tarefas que se encontram no estado de pronto para executar. O desenvolvedor da aplicação deve escolher o algoritmo de escalonamento mais adequado em tempo de projeto. Então, o escalonador selecionado é sintetizado juntamente com a aplicação no código final do sistema embarcado.

2.2.3 API de Comunicação

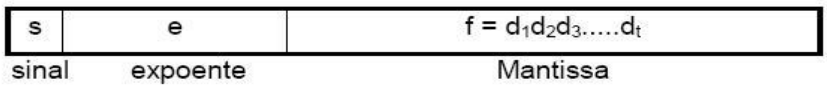
Para prover as facilidades de comunicação foi desenvolvida a APICOM, introduzida em (SILVA JR., 2008a) para funcionar juntamente com o processador FemtoJava, provendo uma interface com a camada de aplicação. A APICOM permite que as aplicações estabeleçam um canal de comunicação através da rede que pode ser usado para prover troca de mensagens entre aplicações que se encontram em processadores diferentes, ou no mesmo processador.

O serviço da API também permite que prioridades diferentes sejam designadas às mensagens, bem como definição de tempos máximos para o envio ou espera de mensagens. Este estabelecimento de tempo é baseado no modelo da RTSJ, integrando as duas APIs. Do ponto de vista da aplicação, o sistema é capaz de abrir e fechar conexões, assim como enviar e receber mensagens, podendo ser acessado por diferentes tarefas simultaneamente. A aplicação ainda pode enviar mensagens no modelo cliente-servidor (orientado a conexão – ponto-a-ponto) ou por *publish-subscribe* (sem conexão prévia – ponto-multiponto).

Assim como no modelo OSI, o sistema de comunicação foi dividido em camadas ou níveis que prestam serviços um para o outro, contribuindo para a melhor comunicação desejável para a aplicação. Nesta API se encontram três camadas: Transporte, Rede e Enlace. Cada uma delas tem funções específicas que complementam as demais.

2.2.4 API Ponto Flutuante

De acordo com o padrão IEEE 754 (ver a Figura 2) recomenda os seguintes números de bits, de acordo com a precisão usada. Meia Precisão: s = 1, e = 5, f = 10 (+ 1 escondido), N = 16 bits. Precisão Simples: s = 1, e = 8, f = 23 (+ 1 escondido), N = 32 bits. A expressão (N = s + e + t) corresponde ao tamanho da palavra em bits.



ISBN 978-85-62830-10-5

Figura 2 - Modelo da representação do ponto flutuante (VIANA, 1999)

Este trabalho usa uma API chamada Softfloat16, que adota o padrão de meia precisão. A API consegue fazer as operações matemáticas básicas de soma, subtração, multiplicação e divisão. Nessa aplicação foram utilizados os métodos de soma e multiplicação para fazer a multiplicação entre matrizes.

2.2.5 SIMPLE

Para verificação do funcionamento de uma aplicação em plataforma FemtoJava, pode-se usar tanto um FPGA, quanto um simulador. Nesse trabalho foi usado o simulador SIMPLE (SILVA JR., 2008b), pela maior facilidade de fazer os experimentos. Ele permite uma simulação em nível de instruções de processador, os bytecodes Java.

O SIMPLE foi desenvolvido em SystemC e pode instanciar um número arbitrário de processadores de acordo com o tamanho da NoC (*System-On-Chip*). Os dispositivos atualmente disponíveis são os temporizadores, as portas de I/O, um relógio de tempo-real e as interfaces de rede.

3. MULTIPLICADOR DE MATRIZES

3.1 Algoritmo

Nessa implementação são dadas duas matrizes **A** e **B**, resultando a multiplicação entre elas, em uma matriz **C**. A multiplicação de matrizes com um processador não utiliza paralelismo. Quando só há um processador operando, ele que multiplica todas as linhas de **A** por todas as colunas de **B**.

Para que haja o paralelismo na multiplicação entre processadores, é preciso estabelecer uma comunicação entre dois ou mais processadores e se faz necessário o uso de um mestre e pelo menos um escravo, como no modelo de memória distribuída. No trabalho foi utilizada a API COM (parte do MiddleSoC). Cada elemento de processamento fica responsável por multiplicar **N** linhas da matriz **A** por todas as colunas da matriz **B**. O mestre deve enviar ao escravo parte da matriz **A** que ele deve multiplicar pela matriz **B**. A matriz **B** também é enviada ao escravo, só que integralmente. Essas matrizes e esses limites de matrizes são enviados por mensagens, depois de estabelecida uma conexão entre eles. Quando o escravo realiza sua parte do algoritmo, ele também deve enviar ao mestre seus resultados para que o mestre monte a matriz C com os valores calculados nos dois processadores.

Esta aplicação foi implementada e simulada e, para verificar o seu funcionamento, o resultado da operação foi pré-calculado em tempo de desenvolvimento e armazenado em uma matriz a parte, chamada **Resultado**. Então, o resultado da multiplicação calculado durante a execução do programa é comparado ao resultado pré-calculado e o programa sinaliza que o cálculo foi bem sucedido.

3.2 Resultados da simulação

Os experimentos de multiplicação de duas matrizes foram feitos com o processador FemtoJava operando a 100MHz. As matrizes **A** e **B** são matrizes trinta e dois por trinta e dois utilizando números reais de meia precisão. Usando dois processadores o código Java do mestre tem 272 linhas, enquanto no escravo possui 253 linhas. Para três processadores o código Java do mestre tem 462 linhas, enquanto no escravo 1 possui 262 linhas e no escravo 2 são 262 linhas.

O tempo gasto pelo multiplicador de matrizes para dois e três processadores realizarem a multiplicação efetiva de matrizes foi de 407,541 ms e 280,182 ms, sendo o tempo gasto com a comunicação entre processadores de 85,448 ms e 127,383 ms. O estabelecimento de conexão, o preparo e o envio de mensagens com as matrizes, a recepção e a transformação de dados recebidos nas mensagens em matrizes novamente fazem parte do custo de comunicação que é proporcionalmente menor que o custo de computação (ver a Figura 3).

A Tabela 1 mostra o custo computacional para execução do mesmo algoritmo de multiplicação de matrizes. Operando em apenas um processador o tempo foi de 816,104 ms. Comparando este

ISBN 978-85-62830-10-5

resultado com os 407,541 ms gastos operando com dois processadores e os 280,182 ms gastos operando com três processadores, se vê o quanto se pode ganhar com o paralelismo. O paralelismo obtido neste caso com apenas dois e três processadores se mostra promissor.

Tabela 1 - Medida de tempo do custo computacional dos processadores

Experimentos	Custo Computacional
Um Processador	*816,104 ms*
Dois Processadores	*407,541ms*
Três Processadores	*280,182 ms*

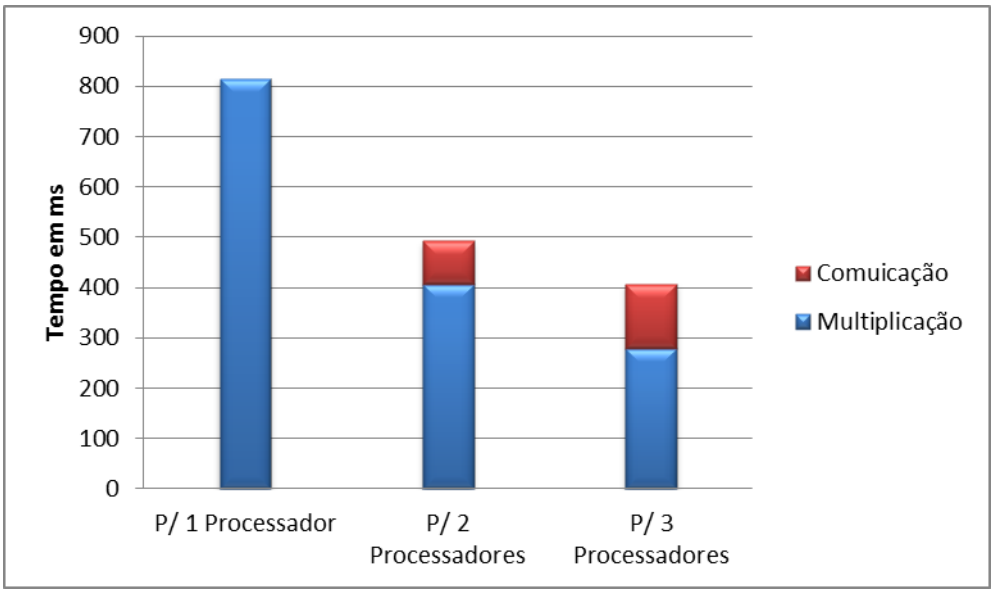


Figura 3 - Custo Computacional

4. Conclusão

Este trabalho apresenta um caso de uso de uma plataforma multiprocessador baseada em uma estrutura de NoC (*Network on Chip*) que utiliza processadores Java. O algoritmo utilizado, multiplicação de matrizes, é base de muitos outros usados em aplicações multimídia, sendo também de fácil paralelização.

Os resultados confirmam que a paralelização reduz sensivelmente o tempo de computação, além de mostrarem que a plataforma utilizada causa um baixo overhead para as operações escolhidas.

Uma proposta de software que também utilize o gerenciamento de energia pode beneficiar sistemas multimídia, quanto ao desempenho e economia de baterias. No caso do multiplicador de matrizes para dois processadores, enquanto o mestre prepara uma mensagem para o escravo, o escravo apenas espera esta mensagem. Adotar uma frequência menor para o processador (neste caso, o escravo) que não estiver operando, já resulta em um ganho de energia. Na continuidade deste trabalho serão usados outros recursos do *middleware*, como o gerenciamento da energia, além de aumentar a quantidade de processadores.

REFERÊNCIAS

BENINI, L.; DEMICHELI, G. **Networks on Chip: A New SoC Paradigm**. In: IEEE COMPUTER, [S.l.], v.35, n.1, p. 490-504, Jan. 2002.

ISBN 978-85-62830-10-5

BOLLELLA, G.; et al. **The Real-Time Specification for Java**. Massachusetts: Addison Wesley Longman, 195 p. 2000.

GILL, C.; CYTRON, R.; SCHMIDT, D.C. **Multi-Paradigm Scheduling for Distributed Real-Time Embedded Computing**. In: PROCEEDINGS OF THE IEEE, PISCATAWAY, SPECIAL ISSUE ON MODELING AND DESIGN OF EMBEDDED SOFTWARE, v.91, n.1, p.183-197, Jan. 2003.

ITO, S.A.; Carro, L.; JACOBI, R.P. **Making Java Work for Microcontroller Applications**. In: IEEE DESIGN & TEST OF COMPUTERS, CALIFORNIA, v.18, n.5, p. 100-110, Sept./Oct. 2001.

SILVA JÚNIOR, E.T.; **Middleware Adaptativo para Sistemas Embarcados e de Tempo-Real**. 2008a. Tese (Doutorado em Ciência da Computação) - Instituto de Informática, UFGRS, Porto Alegre, 2008.

SILVA JÚNIOR, E. T.; BARCELOS, D.; WAGNER, F. R.; PEREIRA, C. E. **A Virtual Platform for Multiprocessor Real-Time Embedded Systems**. In: INTERNATIONAL WORKSHOP ON JAVA TECHNOLOGIES FOR REAL-TIME AND EMBEDDED SYSTEMS, 2008, Santa Clara. JTRES. New York : ACM, 2008b. 31-37 p.

VIANA, G. V. R.; **Padrão IEEE-754 para Aritmética Binária de Ponto Flutuante**. In: CIÊNCIAS E TECNOLOGIA (UECE), FORTALEZA, 1999, v. 1, n. 1, 29-33 p.

WOLF, W. **Computers as Components**. Nova Iorque: McGraw-Hill, 2001.

ZEFERINO, C.A.; SUSIN, A.A. SoCIN: **A parametric and scalable network-onchip**. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, SBCCI, 16., 2003, São Paulo. Proceedings... Los Alamitos: IEEE Computer, 2003. 169-17 p.

ISBN 978-85-62830-10-5