High-Performance Timing Simulation of Embedded Software

Artigo: J. Schnerr, O. Bringmann, A. Viehl, and W. Rosenstiel, "*High-performance timing simulation of embedded software*," in Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE, 2008, pp. 290-295.

Acadêmico: Leonardo Garcia Tampelini – **RA:** 098336

O presente artigo apresentada uma abordagem hibrida para a simulação de precisão de ciclos em softwares embarcados. Em sua proposta, os autores combinam técnicas de simulação e métodos analíticos de avaliação, propondo melhorias na velocidade da simulação sem ocasionar grande perda na precisão dos resultados.

A combinação de componentes mais simples para construção de novos sistemas é apresentada como uma tendência de mercado. Níveis de abstração - causados pela cooperação de componentes - implicam em mudanças no paradigma do projeto, pois a alteração de um único componente pode ocasionar grandes mudanças em nível global. Para avaliar tais situações, os autores enfatizam a necessidade de uma modelagem abrangente, a qual analise e simule a integração do sistema.

Para os autores, as análises estáticas de pior/melhor tempo de execução (WCET/BCET) oferecem resultados pessimistas quando o escopo considerado vai além dos blocos básicos. Este fator é agravado quando os efeitos de coerência de *cache* em arquiteturas *multicore* são considerados. Para reduzir a imprecisão (pessimismo) da analise estática WECT/BCET, a abordagem proposta aplica um esquema de *back-annotation* dos valores WCET/BCET (determinados estaticamente nos blocos básicos do código binários e gerados a partir do código-fonte em C). Além disso, impactos causados por arquiteturas distintas, como diferentes modos de previsão de desvios, poderão ser efetivamente considerados na abordagem proposta.

A abordagem proposta é composta pelas seguintes etapas: **1)** Dado à descrição do processador, traduzir o código binário (utilizando *cross-compiler)* para o processador especificado; **2)** Utilizar a ferramenta de *back-annotation* para ler o código objeto e a descrição do processador para a simulação; **3)** Considerar a descrição do processador para decodificar e traduzir o código objeto em uma representação intermediária (lista de objetos); **4)** A partir da lista de objetos, construir uma lista de blocos básicos do programa; **5)** Utilizar a descrição do *pipeline* do processador (informações capturadas na etapa 2) e calcular estatisticamente o número de ciclos utilizados por cada bloco básico; **6)** Encontrar correspondências entre o código-fonte C e o código binário; **7)** Inserir código para a geração de ciclos que contará os ciclos gastos pelos blocos básicos em sua execução (análise estática); **8)** Finalizar inserindo o código para a correção dinâmica da geração de ciclo. Apesar da arquitetura ser previamente conhecida, sua real influencia no número de ciclos gastos não é, pois certos parâmetros como *cache hit/miss* e previsão de desvios, não podem ser definidos estaticamente, enfatizando a importância da correção dinâmica.

Os testes comparativos foram efetuados utilizando dois filtros (FIR, ellip) e dois programas que fazem parte de rotinas de decodificação de áudio (DPCM, sub-bandas). O código objeto para o *Infineon Tricore* foi gerado em um compilador C. O código objeto foi utilizado para a geração do código *SystemC* anotado. Como referências para medidas foram utilizados o *Tricore TC20GP evaluation board* e o simulador do conjunto de instruções (ISS) Tricore. Foram utilizados dois tipos de anotações no código *SystemC*: **1º)** que gera os ciclos depois da execução de cada bloco básico e **2º)** que adiciona ciclos ao contador de ciclos quando for necessário (comunicação com HW).

Os testes mostram que comparando a velocidade de execução do *SystemC* com a velocidade de execução de um ISS convencional, a abordagem proposta pelos autores promove uma melhora de até 91% na velocidade. Além disso, também foram realizadas comparações da precisão de ciclo (*cycle-accuracy*): o desvio do contador de ciclos dos programas traduzidos (considerando questões de memória e previsão de desvio) em relação ao obtido pelo TC20GP *evaluation bord* variou de 4 a 7% e foi praticamente a mesma variação com relação ao ISS convencional.

Os resultados apresentados neste artigo são promissores, mostrando grande melhora no desempenho e precisão do modelo de software embarcado temporizado (*timed*). Esta abordagem possibilita uma execução rápida do código anotado, além do código *SystemC* gerado poder ser utilizado em ambientes de simulação *SystemC*. Analisar o código binário e o código fonte C pode ser encarada como uma desvantagem dessa abordagem, pois o compilador pode realizar modificações de estrutura e otimização, dificultando a identificação das correspondências, podendo ser necessário utilizar técnicas de recompilação para conseguir identifica-las.