Análise de Valor para Determinação do *WCET* de Tarefas em Sistemas de Tempo Real

Karila Palma Silva
Univ. Fed. Santa Catarina
DAS-CTC-UFSC C.Postal 476
Florianópolis – SC – Brasil
Email: karila.palma@posgrad.ufsc.br

Renan Augusto Starke
Univ. Fed. Santa Catarina
DAS-CTC-UFSC C.Postal 476
Florianópolis – SC – Brasil
Email: renan.xtarke@posgrad.ufsc.br

Rômulo Silva de Oliveira
Univ. Fed. Santa Catarina
DAS-CTC-UFSC C.Postal 476
Florianópolis – SC – Brasil
Email: romulo.deoliveira@ufsc.br

***Resumo*—Neste artigo apresenta-se uma aplicação da análise de valor para determinar a latência das instruções que acessam à memória. O objetivo desta análise é o aprimoramento de ferramentas *WCET*. Análise de valor é utilizada para determinar valores dos registradores do processador estaticamente, possibilitando o reconhecimento dos acessos à memória (memória principal e *ScratchPad Memory*) e a obtenção de um *WCET* mais preciso.**

***Resumo*—This paper presents a method to determine the latency of memory instructions, aimed to use in WCET tools. We use a technique known as value analysis. Value analysis is used to determine the possible values of processor registers statically, allowing the recognition of memory accesses (main memory and Scratchpad Memory) and obtaining tighter WCET.**

I. Introdução

Os sistemas computacionais de tempo real são identificados como sistemas submetidos a requisitos de natureza temporal, onde os resultados devem estar corretos não somente do ponto de vista lógico, mas também devem ser gerados no momento correto. Esses sistemas possuem requisitos que variam com relação ao tamanho, complexidade e criticalidade. As falhas de natureza temporal nesses sistemas são, normalmente consideradas críticas no que diz respeito às suas consequências [1].

No contexto da automação industrial, são muitas as possibilidades de empregar sistemas com requisitos de tempo real, por exemplo, sistemas de controle embutidos em equipamentos industriais, sistemas de supervisão e controle de células de manufatura, sistemas responsáveis pela supervisão e controle de plantas industriais completas, sistemas automotivos, entre outros.

A principal característica de sistemas de tempo real é que eles possuem restrições que aparecem na forma de prazos (*deadlines*). Para o escalonamento de sistemas de tempo real, é necessário que as restrições temporais sejam mapeadas em termos destes prazos. Em testes mais sofisticados, são estabelecidas relações matemáticas mais complexas entre os parâmetros das tarefas. Com a sofisticação dos modelos, a análise torna-se mais complexa, demandando mais recursos computacionais e humanos [1].

Os testes de escalonabilidade provam geralmente a viabilidade da escala determinando se todas as tarefas do sistema podem ou não ser escalonadas cumprindo todos os prazos individualmente. A análise é realizada utilizando os seguintes parâmetros de cada tarefa [2]:

- C_i: *Worst Case Execution Time* – C;
- T_i: período ou tempo mínimo entre chegadas;
- D_i: *deadline.*

Neste contexto, a determinação do tempo de execução no pior caso (*Worst Case Execution Time* (*WCET*)) é um elemento essencial para a análise de escalonabilidade, especialmente em sistemas de tempo real críticos.

Na Figura 1, é ilustrada a problemática da obtenção do *WCET*, o limite esquerdo representa o melhor tempo de execução (*Best Case Execution Time* (*BCET*)), enquanto que o limite direito, o pior tempo de execução (*WCET*). É notória a existência de uma diferença entre esses e as medições de execução. Obter esses valores extremos exatos é muito difícil devido ao grande número de caminhos de execuções possíveis, tornando inviável a exploração exaustiva de todos eles [3].

Para a análise de tempo tornar-se viável, os métodos utilizam abstrações das tarefas, o que causa a perda de informações, e o limite do *WCET* computado normalmente superestima o *WCET* exato enquanto que subestima o *BCET*. O limite do *WCET* representa o pior caso garantido pelo método ou ferramenta utilizado [3].

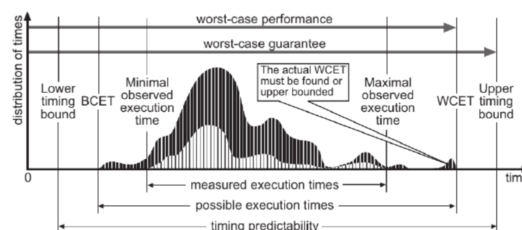

Figura 1. Analise temporal de sistema

Os processadores modernos utilizam técnicas de *pipelining*, *cache* de instruções e dados, *branch prediction* dinâmico, execução fora de ordem, execução especulativa e *multithreading* de granulação fina (instruções de várias *threads* em execução no *pipeline*). O comportamento temporal de um software que executa em processador com

essas técnicas é difícil de modelar para a análise *WCET*. Pretende-se fornecer um tempo de execução seguro mas não extremamente pessimista o que dificultaria sua utilização prática [4], [3].

Sistemas de tempo real precisam de previsibilidade temporal e desempenho de pior caso razoável. Para simplificar a análise do sistema e tornar o pior tempo de execução rápido, é apresentada neste artigo a aplicação de uma técnica de análise de valor, utilizada para o reconhecimento dos acessos à memória (memória principal e *ScratchPad Memory*) com o objetivo de determinar a sua latência. Este reconhecimento permitirá a ferramenta de *WCET* fazer melhores estimativas dos tempos de execução.

II. Análise do *WCET*

Para obter estimativas do tempo de execução, várias técnicas foram desenvolvidas e podem ser classificadas em: estimativas dinâmicas e análise estática. Como as técnicas baseadas em estimativas dinâmicas não garantem que o maior tempo de execução medido seja o verdadeiro tempo de execução no pior caso, o *WCET* pode ocorrer com pouca frequência e as condições para que ele aconteça são normalmente desconhecidas, o foco principal será nas técnicas de análise estática [5].

Na literatura, existe um modelo padrão para análise temporal estática de código, conforme a Figura 2. Esse modelo possui quatro técnicas principais: **reconstrução do fluxo de controle**, que faz a análise do binário da tarefa formando o grafo de controle de fluxo; **análise de valor**, que computa os estados do registradores do processador que serão utilizados na análise da microarquitetura em conjunto com o modelo de hierarquia de memória do sistema; **análise da microarquitetura**, que computa o tempo de execução de cada instrução do processador; e **obtenção do pior caminho**, que após analisar todo o sistema, é formado um problema de otimização com os vários estados obtidos nas análises anteriores, cuja função é encontrar o *WCET* do programa executando naquele modelo de processador e hierarquia de memória [4]. O foco principal deste trabalho é a análise de valor, com o objetivo de obter um *WCET* mais apertado.

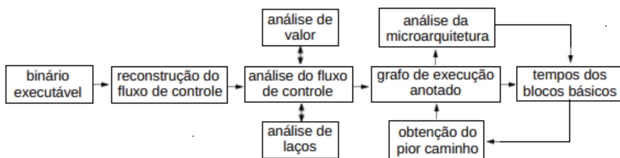

Figura 2. Analise temporal estática de código

III. Análise de valor

A análise de valor é utilizada para determinar o comportamento dos dados e da memória do processador. O endereço efetivo dos acessos à memória está disponível apenas na execução do programa. Dependendo da sua implementação, é possível determinar endereços (computando limites dos valores nos registradores do processador e das variáveis locais a cada ponto do programa) desde que os acessos sejam estáticos e o código gerado pelo compilador seja disciplinado [4].

Esta análise também é útil para determinar limites de iterações de laços e caminhos de execuções inviáveis. Os resultados são utilizados como entrada para análises mais avançadas, o que acaba por produzir informações sobre o uso da pilha e do comportamento temporal [3], [4], [6].

Nesse trabalho a utilização da análise de valor é focada e reduzida. O objetivo é classificar as instruções em acesso a memória principal ou *ScratchPad Memory* calculando a soma da base com o *offset* das instruções de memória. A diferença de acesso tem grande impacto na computação do *WCET*.

IV. Trabalhos relacionados

A análise estática ao nível do código de máquina tem mostrado bons resultados para a estimativa do *WCET*, existem várias abordagens de análise estática para a implementação da análise de valor [3], [7], [4].

Na literatura, a modelagem para a implementação da análise de valor geralmente é realizada pela técnica de interpretação abstrata [8], [9], [10], [3]. Essa técnica é baseada na utilização de uma semântica abstrata para representar os valores durante a execução de um programa, para extrair propriedades de um programa sem que seja necessário executá-lo. É preciso definir as semânticas concretas de forma simplificada que descrevam os aspectos interessantes para a análise de valor, e também definir as semânticas abstratas que coletam os aspectos temporais em cada ponto do programa [7].

A interpretação abstrata é uma técnica promissora, mas para cada processador suportado é necessário construir um modelo seguindo a teoria da interpretação abstrata. É uma solução interessante para analisar um processador comercial, quando seu modelo não está disponível ou já existe uma ferramenta que interprete as semânticas da interpretação abstrata. A implementação deste modelo é uma tarefa complexa exigindo um estudo detalhado da especificação do processador e para obter uma análise confiável é necessário vários passos de validação [7].

Na execução desse trabalho, não foi utilizada técnica de interpretação abstrata para a modelagem da análise de valor. A evolução da especificação do processador no nosso caso acompanha a construção da ferramenta de análise *WCET*, diferente da abordagem padrão que deseja analisar um processador comercial existente.

As ferramentas de *WCET* geralmente reconstroem o fluxo de controle do programa a partir do arquivo executável, analisam as chamadas de funções e a descrição do fluxo a nível de blocos básicos observando apenas o *assembly*. Nesse trabalho o *Control Flow Graph* (*CFG*) [1] é gerado pelo compilador, diminuindo um passo da análise de valor.

[1] *CFG*: é um grafo direcionado $G = (V, E, i)$, onde os vértices V representam os blocos básicos e as arestas $E \subseteq VxV$ conectam dois vértices v_i e v_j se, e somente se v_j é imediatamente executado depois de v_i. O vértice de entrada do *CFG* é representado por i, ou seja, i não possui arestas de entrada $(\neg\exists v \epsilon V : (v, i) \epsilon E)$.

V. Processador e a análise *WCET*

O processador utilizado suporta um subconjunto de instruções da ISA MIPS R200 de 32-bits e possui a seguinte especificação [11]:

Tabela I. Especificação do processador

Recursos	Processador
Pipeline	5 estágios Busca simples Em ordem
Memória	Cache instruções Memória principal *ScratchPad*

Um diagrama simplificado do processador é representado na Figura 3, onde SPRAM é a *Scrachpad Memory* e RAM é a memória principal. A descrição detalhada do modelo do processador está fora do escopo desse trabalho.

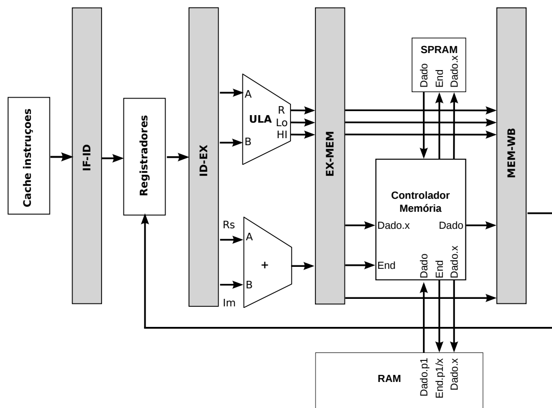

Figura 3. Diagrama simplificado do modelo

O processador foi modelado em SystemC [12] através da metodologia *Register-transfer Level Design* (*RTL*) a nível de ciclo (todo comportamento temporal do *pipeline* e memórias é considerado). Esse mesmo modelo é utilizado para a obtenção do tempo de execução das tarefas e também para a obtenção do tempo de execução dos blocos básicos.

Um ciclo de instrução básico pode ser divido em:

- Busca de instruções (F) - *instruction fetch*;
- Decodificação da instrução (D) - *instruction decode*;
- Execução da instrução (E) - *instruction execution*;
- Acesso à memória (M) - *memory access*;
- Gravação dos resultados (WB) - *write-back*.

Um ciclo genérico busca uma nova instrução da *cache*, decodifica-a determinando que tipo de trabalho será realizado. Geralmente um ou mais operandos devem ser buscados nos registradores. A operação da instrução é executada e o ciclo termina gravando o resultado nos registradores ou na memória [11].

A partir da Figura 3 observa-se a inexistência de uma *cache* de dados com o objetivo de melhorar o determinismo. A *cache* de dados foi substituída pela SPRAM (*ScratchPad Memory*), uma memória rápida porém com conteúdo gerenciado pelo programa e não por *hardware* [11]. Em função dessa característica é necessário classificar os acessos a RAM (memória principal) e a SPRAM durante a análise *WCET*.

As instruções de acesso à memória suportadas são mostradas na Tabela II [11]. Rt e Rs representam o endereço dos registradores codificados na instrução. Im é uma constante de 16-*bits* codificada na instrução.

Tabela II. Instruções de acesso a memória

Mnemônico	Função	Descrição
lw	Rt = MEM[Rs + Im]	Carrega 4-*bytes* da memória (*word*)
lb	Rt = MEM[Rs + Im]	Carrega *byte* da memória
lbu	Rt = MEM[Rs + Im]	Carrega *byte* da memória sem sinal
lh	Rt = MEM[Rs + Im]	Carrega 2-*bytes* da memória (*half-word*)
lhu	Rt = MEM[Rs + Im]	Carrega 2-*bytes* da memória sem sinal (*half-word*)
sb	MEM[Rs + Im] = Rt	Grava *byte* na memória
sh	MEM[Rs + Im] = Rt	Grava 2-*bytes* na memória (*half-word*)
sw	MEM[Rs + Im] = Rt	Grava 4-*bytes* na memória (*word*)

O endereço de acesso é calculado somando a base com o *offset* (Rs + Im). Conforme o valor obtido a instrução é classificada, de acordo com a Tabela III.

Tabela III. Endereçamento de memória e sua penalidade

Memória	Endereçamento (*byte*)	Penalidade (ciclo)
Principal	0 - 0x3FFF	5
ScratchPad	0x4000 - 0x4FFF	0

Dado que o processador possui memória de dados com latências muito diferentes, a classificação correta das instruções é de grande importância para obter um limite superior mais preciso para o *WCET*.

VI. Métodos de classificação e resultados

Nessa seção são apresentados detalhes da implementação, como são classificadas as instruções de acesso a memória, e o resultado da análise *WCET* para alguns *benchmarks*, geralmente utilizados na verificação de ferramentas *WCET*.

A. Implementação

A partir de um arquivo fonte do programa escrito em linguagem C, com a utilização de um compilador (nesse

caso o *Clang+LLVM* estendido para suportar o processador apresentado na Seção V) são produzidos dois dados importantes:

- Objeto (binário) do programa compilado;
- Grafo de fluxo de controle.

Após a compilação, o código objeto passa por uma ferramenta de ligação (*linker*), que é responsável pela leitura das tabelas do código objeto para fazer o mapeamento dos dados e das funções para a configuração de memória do processador.

A análise *WCET* é integrada na mesma ferramenta que o *linker*, diminuindo o fluxo de informações: a análise *WCET* necessita do grafo de fluxo de controle bem como informações sobre o mapeamento de memória do processador.

No grafo de fluxo do programa, cada nó representa um bloco básico [2] que é composto por uma sequência de instruções executadas linearmente sem possibilidade de salto e o limite dos nós são instruções de chamadas, testes ou pulos (*calls*, *branches*, *jumps*).

A nomenclatura dos vértices é formada pelo índice do bloco básico e entre chaves o seu endereço inicial e final. As arestas do grafo indicam o fluxo do programa dos blocos básicos.

Para exemplificar, é mostrado o seguinte código C (*swap.c*):

```
------------------------------
           swap.c
------------------------------

typedef  unsigned char   bool;
typedef  unsigned int    uint;

void inc (uint* a) {
        (*a)++;
}

int main () {
  uint x = 513239;
  inc(&x);
  return 0;
}
```

Observa-se que no *main* do programa *swap.c* tem uma função e sua passagem de parâmetro é por referência. É enviado para a função uma referência da variável utilizada, e não apenas uma cópia.

O *CFG* em conjunto com as instruções de máquina desse programa é mostrado na Figura 4. É possível verificar que no BB 3 o registrador R[29] é inicializado, sendo um ponteiro para a pilha. No BB 0 o R[29] tem um deslocamento de menos 32 e em sequência tem três instruções de acesso a memória que utiliza como base o R[29]. É necessário realizar uma análise de fluxo de dados para a propagação do valor calculado no BB 3 para o seu sucessor (BB 0) e assim sucessivamente.

[2]BB: é uma sequência maximal de instruções, que pode ser alcançado somente pela primeira instrução e a única saída é pela última instrução.

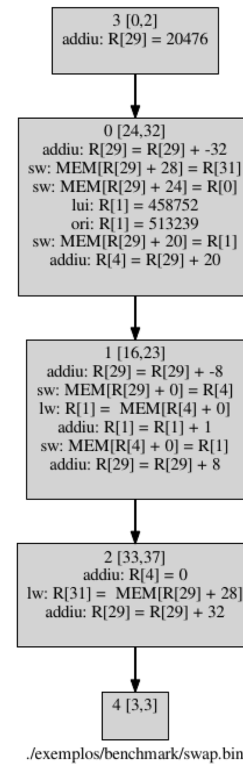

Figura 4. *CFG* com as instruções de máquina do *benchmark swap*

Para a análise do fluxo de dados, utilizou-se o algoritmo *Reaching Memory Blocks* (*RMBs*) [13] ou bloco de memória alcançáveis, onde o conjunto *RMB* contém os estados dos blocos após a execução do bloco básico. O estado de saída dos registradores é copiado dos blocos antecessores e é alterado o estado dos blocos que são acessados internamente.

```
------------------------------------------
Algoritmo de bloco de memória alcançáveis
------------------------------------------
while (change) {
    change = false;

    // Para todos os blocos básico
    for (unsigned int i = 0;
      i < bb_list->size(); i++) {
        basic_block *bb = (*bb_list)[i];

        register_state *pred_state;
        register_state old_state;
        old_state.copy_state(&bb->rmb_rstate);

        if (bb->predecessors.size() == 0) {
            bb->rmb_rstate.copy_state(&bb->my_rstate);
        }

        // Para todos os predecessores
        for (unsigned int j = 0;
          j < bb->predecessors.size(); j++) {
            basic_block *pred = bb->predecessors[j];

            pred_state = new register_state;
            pred_state->last_value(0, 0);

            // Método das instruções
            rmb_instruction(bb, pred, pred_state);

            delete(pred_state);
        }

        // Iterar até convergir
        change = change |
          !(old_state.is_egual(&bb->rmb_rstate));

        bb->rmb_rstate.print();
    }
}
```

O método *rmb_instruction* é responsável pela interpretação estática das instruções, respeitando as seguintes

condições:

- Zerar o registrador destino da instrução de leitura de memória (*lw*, *lb*, *lh*, *lhu*), pois o conteúdo da memória não é considerado nesta análise.
- Instruções que não acessam a memória são interpretadas apenas quando seus parâmetros (Rs, Rt) são conhecidos.

Primeiramente são calculados os valores dos registradores em cada iteração do programa *swap* com a utilização do *RMBs* propagando apenas as constantes do programa, ou seja, sem considerar as instruções de acesso a memória, o resultado é apresentado na Tabela IV.

Tabela IV. RMB - PROPAGAÇÃO DAS CONSTANTES

Iteração	BB: 0	BB: 1	BB: 2
1	R[0]: 0 R[1]: 513239	R[0]: 0 R[1]: 513240	R[0]: 0 R[1]: 513240 R[4]: 0
2	R[0]: 0 R[1]: 513239 R[4]: 20464 R[29]: 20444	R[1]: 513240 R[4]: 20464 R[29]: 20444	R[1]: 513240 R[4]: 0 R[29]: 20476

Iteração	BB: 3	BB: 4
1	R[0]: 0 R[29]: 20476	R[0]: 0 R[1]: 513240 R[4]: 0
2	R[0]: 0 R[29]: 20476	R[0]: 0 R[1]: 513240 R[4]: 0 R[29]: 20476

Após realizar a propagação das constantes, considera-se todas as instruções. Os valores resultantes dos registradores calculados do programa *swap* são apresentados na Tabela V.

Tabela V. RMB - TODAS AS INSTRUÇÕES

BB: 0	BB: 1	BB: 2
R[0]: 0 R[1]: 513239 R[4]: 20464 R[29]: 20444	R[29]: 20444	R[4]: 0 R[29]: 20476

BB: 3	BB: 4
R[0]: 0 R[29]: 20476	R[0]: 0 R[4]: 0 R[29]: 20476

Observando o *CFG* com as instruções de máquina do *benchmark swap*, Figura 4, e os valores resultantes dos registradores calculados com o *RMBs*, Tabelas IV e V, é possível verificar que o registrador destino de uma instrução de acesso a memória não é computado, devido as condições assumidas. Por exemplo, no BB 1 da Tabela IV tem que R[1] = 513240 e na Tabela V o R[1] do BB 1 não possui valor, pois tem a instrução LW: R[1] = MEM[R[4]+0], desta forma, o R[1] desse bloco básico é zerado e não é possível computar a instrução ADDIU: R[1] = R[1]+1. O que nesse caso não é um problema, pois R[1] não é base para nenhuma instrução de memória.

Para a classificação das instruções de memória o endereço de acesso é calculado. A instrução é classificada em *SLOW* ou *FAST*, Tabela VI:

Tabela VI. CLASSIFICAÇÃO DA MEMÓRIA

Classificação	Memória	Endereçamento (decimal)
SLOW	Principal	0 - 16383
FAST	*ScratchPad*	16384 - 20479

Para exemplificar as Tabelas VII, VIII, IX ilustram a classificação das instruções de acesso a memória do programa *swap*.

Tabela VII. CLASSIFICAÇÃO BB:0

BB: 0	
Instrução	Classificação
SW: Mem[20472] = R[31]	FAST
SW: Mem[20468] = R[0]	FAST
SW: Mem[20464] = R[1]	FAST

Tabela VIII. CLASSIFICAÇÃO BB:1

BB: 1	
Instrução	Classificação
SW: Mem[20436] = R[4]	FAST
LW: R[1] = Mem[20464]	FAST
SW: Mem[20464] = R[1]	FAST

Tabela IX. CLASSIFICAÇÃO BB:2

BB: 2	
Instrução	Classificação
LW: R[31]: Mem[20472]	FAST

Todas as instruções de acesso à memória do programa *swap* foram classificadas como *FAST*, então possuem latência determinada pela *ScratchPad Memory*.

Sem a análise de valor, todas as instruções de acesso à memória seriam classificadas como *SLOW*. Nesse caso, o valor do *WCET* dos programas seria superestimado consideravelmente.

Conforme apresentado, com a análise de valor implementada é possível garantir que a classificação de todas as instruções de memória do programa *swap* é *FAST*, obtendo um limite superior do *WCET* mais preciso.

B. *WCET*

Para a verificação da ferramenta desenvolvida, utilizando alguns *benchmarks* de análise do *WCET* [14], considerou-se a análise de *WCET* sem a análise de valor (qualquer instrução de acesso à memória possuem latência determinada pela memória principal), e a análise de *WCET* com a análise de valor implementada. A relação (R_1) dessas análises podem ser visualizadas na Tabela X. A relação (R_1) é dada pela Equação 1.

$$R_1 = (semAV) - (comAV) \quad (1)$$

AV: análise de valor.

Tabela X. Análise do *WCET* com e sem análise de valor

Benchmark	*WCET* com AV	*WCET* sem AV	R_1
break2	172	418	246
bs	232	526	294
bsort100	924669	1891953	967284
cnt	8963	21113	12150
cover	5834	12452	6618
crc	149437	370231	220794
edn	338775	831993	493218
fdct	7222	18940	11718
fibcall	209	713	504
insertsort	6029	10325	4296
janne complex	3537	9417	5880
lcdnum	716	1574	858
matmult	582729	1064715	481986
ns	25815	60501	34686
prime	13514	37010	23496
swap	31	73	42

Observa-se que a análise de *WCET* com a análise de valor diminuiu significativamente o número de ciclos.

Na Tabela XI, mostra-se para cada *benchmark* o *WCET* obtido considerando a análise de valor, o tempo de execução (SIM), e a relação (R_2) entre esses. A relação (R_2) é dada pela Equação 1.

$$R_2 = \left(1 - \frac{SIM}{WCET} \right) * 100 \tag{2}$$

Tabela XI. Análise do *WCET*

Benchmark	*WCET* com AV	SIM	R_2
break2	172	172	0
bs	232	226	2,6
bsort100	924669	460830	50,2
cnt	8963	8963	0
cover	5834	5834	0
crc	149437	74082	50,4
edn	338775	337286	0,4
fdct	7222	7222	0
fibcall	209	209	0
insertsort	6029	3577	40,7
janne complex	3537	544	84,6
lcdnum	716	486	32,1
matmult	582729	582729	0
ns	25815	25809	0
prime	13514	13496	0,1
swap	31	31	0

Conforme os resultados apresentados da análise dos *benchmarks*, verifica-se que existem casos onde há uma grande diferença entre o pior tempo de execução e a execução real do programa, essa diferença ocorre normalmente quando o número máximo de iterações dos laços é inferior à execução real.

VII. Conclusão

A análise de sistemas de tempo real é uma tarefa desafiadora e requer a união de várias técnicas, para a obtenção analítica do pior tempo de execução de tarefas, uma etapa importante no desenvolvimento e validação de sistemas de tempo real críticos. Os valores estimados ou calculados do pior tempo de execução podem ser utilizados para verificar: a análise de escalonamento, se as tarefas periódicas satisfazem seus objetivos de desempenho, se as interrupções possuem tempos de reação suficientemente curtos, gargalos de desempenho, entre outros [5].

A contribuição mais relevante deste trabalho é a classificação das instruções (acesso a memória principal ou *ScratchPad Memory*). A partir da reconstrução do fluxo do controle do programa são formados os blocos básicos de execução e é realizado um mapeamento dos acessos aos segmentos de memória, possibilitando classificar as instruções e gerar um limite superior do *WCET* mais apertado. Como a penalidade de acesso à memória principal é cinco vezes superior, a classificação correta das instruções tem grande impacto no *WCET*.

Em *benchmarks* onde o fluxo e número de iterações são comportados a análise *WCET* apresentou resultados precisos devido ao comportamento determinista do processador e ao fluxo comportado do programa.

A ferramenta desenvolvida pode ser aprimorada para reduzir os resultados pessimistas. Estão sendo estudadas outras formas para diminuir aproximações e o resultado da análise ser o mais próximo do tempo de execução real do programa.

Referências

[1] J.-M. Farines, J. da Silva Fraga, and R. S. de Oliveira, "Sistemas de Tempo Real," *12a Escola de Computação*, 2000.

[2] G. C. Buttazzo, "Hard Real Time Computing Systems: Predictable Scheduling Algorithms and Applications," *Vasa*, 2008.

[3] R. Wilhelm, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, P. Stenström, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, and R. Heckmann, "The worst-case execution-time problem overview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 3, Apr. 2008.

[4] R. A. Starke, "Uma Abordagem de escalonamento heterogêneo preemptivo e não preemptivo para sistemas de tempo real com garantia em multiprocessadores," Master's thesis, Universidade Federal de Santa Catarina, 2012.

[5] G. Alvarez, "Caracterização Analítica de Carga de Trabalho Baseada em Cenários de Aplicações Multimídia," Ph.D. dissertation, Universidade de São Paulo, 2013.

[6] R. Heckmann and C. Ferdinand, "Verifying safety-critical timing and memory-usage properties of embedded software by abstract interpretation," *Automation and Test in Europe-Volume 1*, no. IEEE Computer Society, pp. 618—-619, 2005.

[7] S. Thesing, "Safe and Precise WCET Determination by Abstract Interpretation of Pipeline Models Dissertation," Ph.D. dissertation, Universität des Saarlandes, 2004.

[8] H. Cassé, F. Birée, and P. Sainrat, "Multi-architecture value analysis for machine code," in *OASIcs-OpenAccess Series in Informatics*, vol. 30. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[9] H. H. Harrison, "Compiler analysis of the value ranges for variables," *Software Engineering, IEEE Transactions on*, no. 3, pp. 243–250, 1977.

[10] Y. Markovskiy, "Range Analysis with Abstract Interpretation," *Semester Project, CS*, no. 13099223, pp. 1–8, 2002.

[11] D. A. Patterson and J. L. Hennessy, *Computer organization and design: the hardware/software interface*. Newnes, 2013.

[12] A. S. Initiative, "Systemc," <http://www.accellera.org/home/>, 2013.

[13] R. A. Starke and R. S. de Oliveira, "Cache preemption related delay accounting via static analysis and functional simulation," in *Computing System Engineering (SBESC), 2012 Brazilian Symposium on*. IEEE, 2012, pp. 149–152.

[14] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper, "The Mälardalen WCET benchmarks – past, present and future," B. Lisper, Ed. Brussels, Belgium: OCG, Jul. 2010, pp. 137–147.