Interpretação Binária do Código x86 em um Processador x86 *

Marconi Rivello, Edil S. T. Fernandes

Relatório Técnico ES-625/03
Programa de Engenharia de Sistemas e Computação
COPPE—UFRJ
Cx. Postal 68.511
21945-970 – Rio de Janeiro – RJ, Brasil
marconi, edil@cos.ufrj.br

Rio de Janeiro, 22 de dezembro de 2003

Resumo

Este relatório descreve o desenvolvimento de um **Interpretador Binário** alternativo do código x86 em um processador x86. Nosso processo de interpretação emprega o código binário de um executável e a correspondente estrutura de dados, que foi gerada anteriormente por um *Disassembler*, que irá controlar o processo.

A vantagem oferecida por nosso Interpretador consiste no reduzido tempo de processamento requerido. Ao invés de interromper o processo após a interpretação de cada instrução do x86, nosso interpretador continua tratando as instruções subseqüentes e somente na fronteira do bloco básico corrente é que ele detecta que é necessário interromper a interpretação. Em seguida, restauramos o código original do binário (que havíamos substituído anteriormente e que provocou a presente interrupção) e passamos para a interpretação do bloco básico sucessor.

A injeção (i.e., *binary rewriting*) de pontos de parada sobre a última instrução de cada bloco básico não é realizada se ele incluir menos que três instruções: copiar a última instrução do bloco básico, substituí-la por um ponto de parada e restaurá-la quando o fluxo de controle atingir aquele *breakpoint*, apresenta um custo que não compensa se o bloco básico contiver menos que três instruções.

Interpretação Binária é uma poderosa ferramenta que auxilia as fases de concepção de novas arquiteturas de computador, de geração e de otimização de código. Nosso interpretador alternativo já foi validado e estatísticas de desempenho foram coletadas. Os resultados produzidos por nossos experimentos indicam que o processo alternativo de interpretação de blocos básicos apresenta um ganho de desempenho de 60% quando comparado com o processo que trata uma única instrução por ciclo de interpretação.

*Pesquisa financiada pelo CNPq – Projeto 55.2091/02-2

1 Introdução

Neste trabalho realizaremos a revisão de duas técnicas para a interpretação controlada de executáveis x86 em um processador x86. A primeira técnica consiste em interpretar individualmente cada instrução do binário x86 conforme descrita em [FREF03]. Embora viável, esta técnica é ineficiente devido ao grande *overhead* provocado pelas constantes trocas de contexto exigidas pelo processo de interpretação. Uma segunda técnica, derivada da primeira, considera o bloco básico como unidade padrão de interpretação. Conseqüentemente, ela evita muitas trocas de contexto envolvendo o processo monitor e o monitorado.

Uma vez que estamos interessados em interpretar binários x86 de forma controlada (i.e., durante a interpretação queremos acessar a próxima instrução a ser interpretada e seus operandos, o conteúdo de registradores e de qualquer posição de memória pertencente ao processo monitorado), a interpretação então será levada a cabo por um **processo monitor** que, após seu início, realiza um *fork()*, gerando assim um processo filho (i.e., o **processo monitorado**). O **monitor** copia a imagem do executável que se deseja monitorar, através da função `execl()` do Linux. Neste momento, o processo filho é uma instância do executável monitorado, e fica suspenso. O monitor possui, então, controle total sobre a memória, os registradores, e a execução do processo monitorado, através de chamadas ao `ptrace`.

A grande vantagem das duas técnicas de interpretação que estamos investigando é que elas não contaminam o código executável com a inclusão de instruções de monitoração.[1] Nos nossos métodos ocorre apenas a troca (temporária) do código de operação da instrução que deve interromper temporariamente o processo de interpretação por um *breakpoint* (como faz o *gnu debugger*). Após atingir o *breakpoint*, o processo monitorado é interrompido e o código de operação da instrução original é restaurado.

O restante deste relatório está assim organizado: o Capítulo 2 descreve as estruturas requeridas pelo interpretador; apresentamos o meio ambiente experimental que usamos no Capítulo 3 e os experimentos que foram realizados no Capítulo 4; no Capítulo 5 reproduzimos os resultados obtidos nos experimentos; e concluímos o relatório com o Capítulo 6.

2 Meio Ambiente Experimental

Nesta fase do projeto BINTRAN estamos interessados em reproduzir o comportamento da execução de binários do x86 de forma controlada. A desvantagem do procediemnto é o elevado tempo de processamento requerido pela interpretação: ocorre uma perda de desempenho na reprodução do comportamento e este trabalho visa reduzir esta perda.

A máquina usada nos experimentos (nossa máquina hospedeira) foi um Intel Pentium 4, com Sistema Operacional Linux. A *target machine* (máquina alvo), ou seja, o código de máquina que deve ser intrepetado, é um executável da máquina x86.

[1] A inclusão de instruções adicionais contamina o código porque a reprodução do ambiente de execução deixa de ser fidegina: *instruction cache misses* é um exemplo típico de distúrbio causado pela inclusão de instruções extras.

O Linux foi escolhido por diversos motivos, como estabilidade, flexibilidade, possuir vasta documentação e por ser do tipo *software* livre.

Os executáveis investigados durante os testes foram os programas inteiros contidos no pacote SPEC 2000, por ser um *benchmark* amplamente utilizado na área de arquiteturas de computadores. Estes programas foram compilados pelo GCC 2.96 com as rotinas da biblioteca `GLIBC` ligadas estaticamente e o conjunto de dados de entrada "*reference*" (também fornecido na distribuição SPEC 2000) foi empregado.

3 Metodologia

Recentemente, o tópico **Interpretação/Tradução Binária** foi revigorado com o aparecimento de novos processadores que empregam técnicas de Tradução e Interpretação para atingir elevados níveis de desempenho no processamento de antigos programas objeto especificados no repertório de instruções (ISA) da tradicional família de processadores da Intel (o x86).

O tópico Interpretação pode ser visto como uma vertente de trabalhos conduzidos na década de 60 do século passado em microprogramação. Dentre estes trabalhos podemos destacar a implementação microprogramada da linguagem de alto nível Euler, que foi levada a cabo por H. Weber em um computador IBM da série System/360 (vide [HWEB67]). Neste trabalho, o autor descreve o processo de interpretação como formado por uma

3.1 PTrace

Para monitorar um processo, usaremos o ptrace. Ptrace é uma chamada de sistema que permite um processo controlar a execução de outro. Também permite que um processo altere dados da memória de outro. O processo monitorado se comporta normalmente até que receba um sinal do sistema operacional. Quando isso ocorre, o processo monitorado fica parado, e o processo monitor é informado através da chamada wait(). Então, o processo monitor decide como o processo monitorado deve se comportar.

Ao fazer uma chamada ao *ptrace*, o processo deve passar como parâmetro uma requisição, e o identificador (pid) do processo a ser monitorado. Algumas requisições utilizam dois outros parâmetros: *addr* e *data*. A seguir, descreveremos as requisições mais importantes:

- PTRACE_TRACEME: Indica que este processo deve ser monitorado. Ao receber qualquer sinal, sua execução será interrompida e seu monitor será avisado através da chamada *wait()*.

A requisição acima é usada apenas pelo processo monitorado. As demais só podem ser utilizadas por um processo monitor, e devem indicar através do parâmetro *pid* em qual processo deve ser feita a requisição. Para qualquer requisição que não seja *PTRACE_KILL*, o processo monitorado deve estar parado.

- PTRACE_PEEKTEXT: Lê uma palavra na posição de memória *addr* do processo monitorado, retornando a palavra como resultado da chamada ao *ptrace*.

- PTRACE_POKETEXT: Copia a palavra *data* para a posição *addr* da memória do processo monitorado.
- PTRACE_GETREGS, PTRACE_GETFPREGS: Copia os registradores de uso geral, e os registradores de ponto flutuante, respectivamente, para a posição *data* de memória do processo monitor.
- PTRACE_SETREGS, PTRACE_SETFPREGS: Copia, para o processo monitorado, os registradores de uso geral, e os registradores de ponto flutuante, respectivamente, da posição *data* da memória do processo monitor.
- PTRACE_CONT: Continua a execução do processo monitorado, até que haja um sinal.
- PTRACE_SINGLESTEP: Continua a execução do processo monitorado, como em *PTRACE_CONT*, mas para o processo monitorado após a execução de uma única instrução.

Através dessas requisições, é possível monitorar um executável x86, e controlar sua execução.

3.2 Estruturas do Interpretador

No interpretador, é usado como entrada um arquivo gerado previamente (vide [FREF03]) por um *disassembler*. Esse arquivo identifica as instruções, início e fim de blocos básicos. Essa informação é necessária porque existem, além das instruções, outros dados inseridos no arquivo executável.

O *disassembler* gera um arquivo que descreve cada *byte* do executável. Para cada *byte* do executável existe um registro correspondente que indica, entre outras coisas, se o *byte* pertence a uma instrução, se é o primeiro *byte* de um bloco básico e se é o último *byte* de um bloco básico. Essas informações serão cruciais para o algoritmo desenvolvido.

O arquivo gerado pelo *disassembler* descreve a seção de texto do binário. O Linux carrega a seção de texto na posição virtual de memória 0x080480e0. Então, o primeiro registro do arquivo do *disassembler* é referente ao primeiro byte da seção de texto do programa.

Cada registro é constituído por 10 *bytes*, sendo 2 para armazenamento dos *flags* descritos a seguir, e 8 para armazenar o contador de quantas vezes (se aplicável) a instrução correspondente foi executada. Embora presente no arquivo, esse contador não possui uso no monitor.

Flags:
- se o *byte* é início de instrução (0x0004)
- se o *byte* é o último de uma instrução (0x0010)
- se o *byte* é início de um bloco básico (0x0020)
- se o *byte* é o último de um bloco básico (0x0040)

4 Técnicas de Interpretação

Usaremos o *ptrace* para monitorar a execução controlada de um binário x86 no nosso ambiente. Nas duas técnicas que descreveremos a seguir, utilizamos dois processos: o monitor, e o monitorado. O programa monitor é executado, e se

auto-duplica através da chamada *fork()*. A partir desse ponto, há dois processos inicialmente iguais: o processo pai (o monitor) e o filho (processo monitorado). O processo filho substitui sua imagem pela do binário que se deseja executar, através da chamada *execl()*, e sua execução é interrompida. O processo monitor faz uma requisição ao *ptrace* com a requisição *PTRACE_ATTACH*, e a partir deste ponto, o processo filho já pode ser monitorado, inspecionado e alterado pelo processo pai.

Para executar o processo monitorado de forma controlada, podemos utilizar duas técnicas que descrevemos a seguir.

4.1 Interpretação Passo a Passo

O processo monitor pode executar o processo monitorado passo a passo, permitindo a execução de uma única instrução por vez, através da requisição *PTRACE_SINGLESTEP* ao *ptrace*. Dessa forma, o processo monitor chama o *ptrace*, que coloca o processo monitorado de volta na fila de execução do linux. O processo monitor, então, chama a rotina *wait()*, que só retornará quando o processo monitorado for novamente interrompido. Nesse instante, há uma troca de contexto entre o processo monitor e o monitorado. O que, na verdade, o *ptrace* fez com o processo monitorado, foi habilitar seu *flag* de *trap*, e recolocá-lo na fila de execução. Após a execução da instrução corrente, o *trap* gerará um sinal para o processo, que interromperá sua execução, ocasionando uma nova troca de contexto entre o processo monitorado e o monitor.

Assim, podemos executar o programa desejado, e em qualquer ponto de sua execução inspecionar suas instruções, seus registradores e sua memória.

O principal problema dessa técnica é o alto número de trocas de contexto. O processo monitorado só executa uma única instrução cada vez que obtém o controle da CPU. O número de instruções executadas para realizar essas trocas de contexto é muito superior ao número de instruções executadas pelo próprio programa.

4.2 Interpretação por Blocos Básicos

Para reduzir o elevadíssimo número de trocas de contexto entre o processo monitor e o monitorado, precisamos garantir que o processo monitorado seja capaz de executar mais de uma instrução toda vez que ele obtiver o controle da CPU. Infelizmente, o *ptrace* não permite especificar o número de instruções a serem executadas. Contudo, ele permite que o programa seja executado até que o mesmo receba um sinal, possivelmente gerado por um *trap*.

Existe, no x86, uma instrução que gera um *trap*. Esta instrução é a *Int3* e seu código hexadecimal é *0xCC*. Ao invés de fazer a requisição *PTRACE_SINGLESTEP*, podemos empregar *PTRACE_CONT*, que continuará a execução do programa até que ocorra um *trap*.

Não é possível incluir *traps* a cada, digamos, duas instruções do programa objeto porque isso modificaria os endereços das instruções do programa, exigindo a alteração de todos endereços de desvio, o que é, em muitas das vezes, impossível. Além disso, instruções de desvio entre *traps* podem fazer com que sejam executadas mais instruções do que o esperado. Inserir traps ao longo do programa "contaminaria´´ o código do executável, possivelmente gerando um comportamento inesperado.

Utilizaremos o conceito de bloco básico para poder executar mais de uma instrução antes de retornar o controle ao monitor. Um bloco básico é uma sequência de instruções que ou é executada por completo, ou nenhuma de suas instruções é executada. Ou seja, não há instruções de desvio, nem desvios para uma instrução da sequência que não seja a primeira.

Para executar blocos básicos, ao invés de uma única instrução, utilizamos as informações contidas no arquivo gerado pelo *disassembler*. Esse arquivo nos fornece os endereços de início e fim de blocos básicos.

Ao iniciar a execução do processo monitorado, o monitor substitui a última instrução do bloco básico em que se encontra por um *trap*, e faz uma chamada ao *ptrace* com a requisição *PTRACE_CONT*, que executará o processo monitorado até que o mesmo enconter um *trap*. Ao retomar o controle, o processo monitor deve restaurar a última instrução do bloco básico, e executá-la através da chamada *PTRACE_SINGLESTEP*. Neste momento, o processo monitorado se encontra no início de um outro bloco básico, o que permite continuar sua execução por blocos básicos.

Descreveremos o algoritmo de execução por blocos básicos a seguir:

```
salva a última instrução do bloco básico
substitui a original pela instrução Int3 (0xCC)
continua a execução do processo monitorado
aguarda a interrupção do processo monitorado
restaura a instrução original, previamente salva
volta o intruction pointer para a última instrução do bloco básico
executa em single step a última instrução
```

O algoritmo acima é executado cada vez que o monitor identifica o início de um bloco básico conhecido (no arquivo gerado pelo *disassembler*). O processo monitor aguarda a interrupção do monitorado através da função *wait()*. Ao ser executada, a instrução *Int3* gera um *trap*, transferindo o controle de volta ao processo monitor. A alteração do *EIP* e execução em *single step* é necessária porque a instrução *Int3* foi executada no lugar da última instrução do bloco básico. Após a execução da instrução restaurada, o monitor verifica se a instrução seguinte é a primeira de um bloco básico conhecido e reinicia o processo.

5 Resultados

Utilizamos os programas inteiros, contido no pacote *SPEC 2000*, para comparar a execução controlada de um processo utilizando as duas formas descritas, ou seja, em *single step* e por blocos básicos. O conjunto de entradas utilizado foi o *test*.

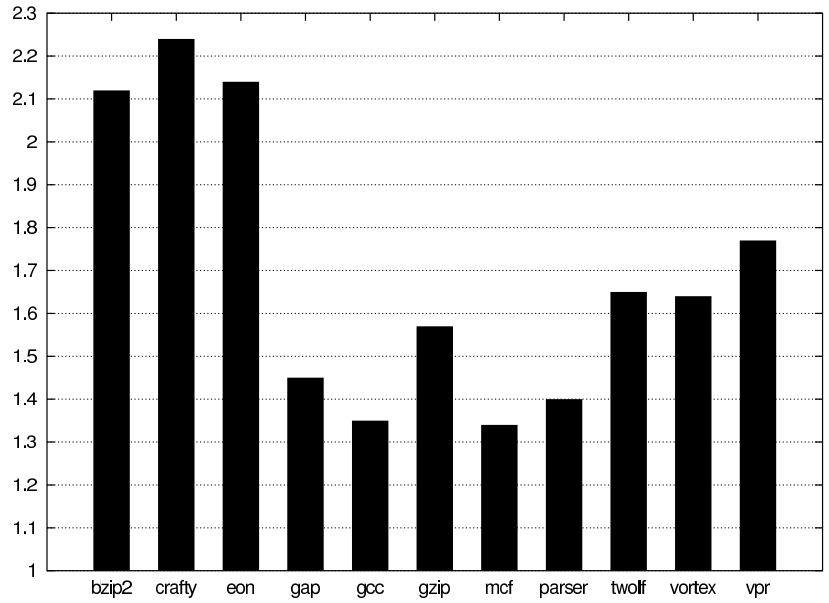
A seguir, apresentamos a tabela com os dados obtidos pela interpretação dos programas inteiros do SPEC 2000.

Prog.	Tss	Tbb	IC	SS	TT	BB
bzip2	38709,860727	18280,972227	8285419933	259892659	1017022209	1162779477
crafty	24883,899749	11126,019967	5349246299	320929833	558154028	759117760
eon	10516,203820	4923,740587	2226460591	66121086	268855173	321464327
gap	4126,788858	2848,793327	872946872	130739105	124606919	198080941
gcc	7156,721954	5292,869255	1526340660	222201035	242059676	375182335
gzip	11391,962390	7235,544654	2470942007	296985556	336780027	516272392
mcf	912,174101	680,814429	193184949	29386243	30687724	47178093
parser	15448,204909	11014,044136	3328755857	254218169	591854839	747765055
twolf	1076,312810	651,059430	227744921	19471504	32262817	44699264
vortex	41399,726248	25195,610246	8812973339	901030001	1206262964	1863968811
vpr	7457,807438	4204,775892	1565243597	227800635	169854826	295844915

Legenda	
Prog.	Nome do programa interpretado
Tss	Tempo de interpretação em *single step*
Tbb	Tempo de interpretação por blocos básicos
IC	*Instruction count* - Número de instruções interpretadas
SS	Número de instr. interpretadas em *single step* na interpret. por b.básico
TT	Número de blocos básicos aproveitados pelo método (*traced to*)
BB	Número de blocos básicos encontrados (dinâmico)

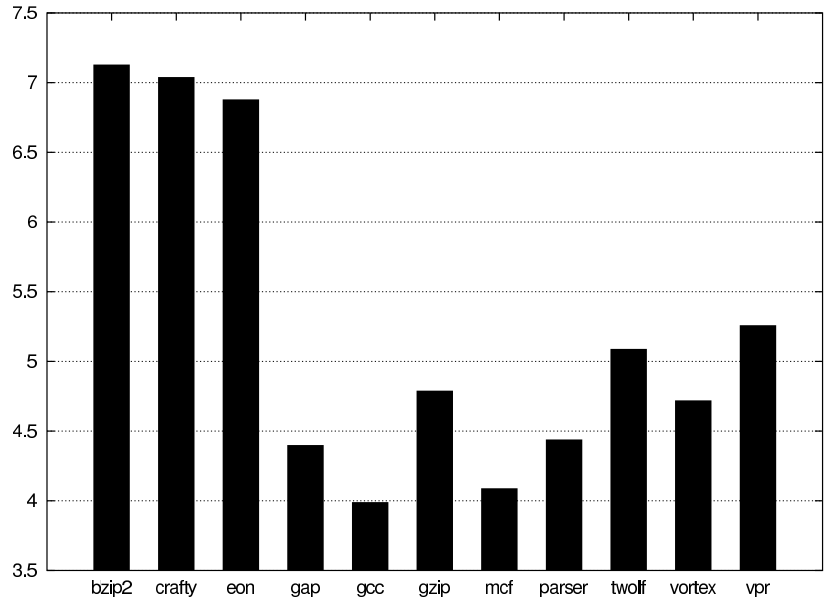
Apresentamos em seguida o gráfico comparativo com o número de instruções de cada programa de teste que foram interpretados.

Instruções x86 Interpretadas



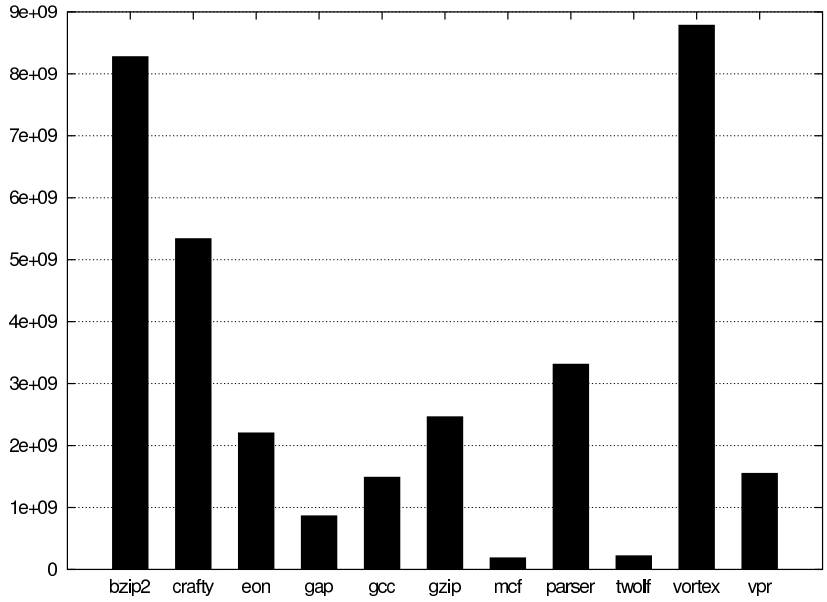
O gráfico abaixo mostra o tamanho médio dos blocos básicos de cada *benchmark*.

Tamanho Médio dos Blocos Básicos



A seguir, apresentamos o gráfico de ganho de performance da técnica de execução por blocos básicos em relação ao *single step*. Observa-se a relação direta entre o tamanho médio dos blocos básicos e o ganho de performance.

Gráfico Comparativo de Desempenho (*Speedup*)



6 Conclusões

Foram desenvolvidos dois programas monitores, para a execução (i.e., intepretação) controlada de um executável x86 num processador x86. Utilizando a estrutura de dados gerada pelo *disassembler* descrito em [FREF03], desenvolvemos um processo alternativo de interpretação binária que evita muitas trocas de contexto entre o processo monitor e o monitorado, reduzindo significativamente o tempo de interpretação.

Os resultados produzidos pelos experimentos mostram ganhos de no mínimo 34%, máximo de 124% e com um *speedup* médio de 70%. Estes ganhos são significativos e comprovam que nosso processo alternativo de interpretação binária é bastante vantajoso.

Mesmo assim, ainda existe um *overhead* gerado pela troca de contexto entre o *kernel* do Linux e o processo monitor. Nosso próximo passo será modificar o *kernel* e a função `Ptrace` para reduzir, mais ainda, este *overhead*. Nossa investigação prossegue.

7 Referências

[DUCC00] D. Ung, and C. Cifuentes, "Machine - adaptable Dynamic Binary Translation," Proceedings of the ACM Workshop on Dynamic Optimization, Dynamo '00, 2000.

[FREF03] Fabiano Ramos e Edil S T Fernandes, "*Disassembly* — Uma Fase Essencial para a Tradução e Interpretação Confiável de Binários," Relatório Técnico ES-624/03, Programa de Sistemas e Computação, COPPE/UFRJ, 22 de dezembro de 2003, 11 pgs.

[HWEB67] Helmut Weber, "A microprogrammed Implementation of EULER on IBM System/360 Model 30," C ACM, Vol. 10, No. 9, September 1967.

[LHSU97] Liangchuan Hsu, "A Robust Foundation for Binary Translation of x86 Code," Ph.D. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1997.