A Database for Reproducible Computational Research

Gabriel F. T. Gomes, Edson Borin
Universidade Estadual de Campinas
Instituto de Computação
Av. Albert Einstein, 1251, Campinas, Brasil
gabriel.ferreira@lsc.ic.unicamp.br, edson@ic.unicamp.br

***Abstract*—The ability to reproduce the experiments of a scientific research is one of the fundamental principles of the scientific method, as failing to reproduce it and obtain results equal (or very similar) to the original might imply that the later were wrong. In compiler and computer architecture research, reproducibility requires a researcher to be able to compile and execute software on (possibly many) computing systems in a repeatable manner. Though the source code for the studied applications, operating systems and compiler toolchains are often available, it is sometimes still impossible to exactly reproduce one experiment due to the lack of information about versioning of the code, build configuration and environmental setup. In this work we propose a database for gathering experimental setup information as well as related experimental results to enforce reproducibility and to enable results sharing among the researchers.**

***Keywords*-database; reproducibility; scientific method**

I. Introdução

De acordo com a Sociedade Americana de Física [1], "ciência é a atividade de coleta sistemática de conhecimento sobre o universo e a condensação desse conhecimento em leis e teorias. O sucesso e a credibilidade da ciência dependem da boa vontade dos pesquisadores exporem as suas ideias e resultados para testes e reprodução independentes, por outros cientistas" [2]. A verificação da validade dos resultados de pesquisas científicas permite a detecção de possíveis erros [3], [4] e fraudes [5].

Diversas áreas da ciência enfrentam os problemas relacionados à determinação da validade das suas descobertas [3], [4], [6]. Nas ciências naturais, a tradição diz que os resultados de um experimento científico devem ser passíveis de reprodução por pesquisadores terceiros em qualquer parte do mundo [6], [7]. Como exemplo podemos citar o esforço da comunidade de pesquisas sobre genoma que se reuniu em 1996 nas Bermudas para desenvolver um método unificado de análise e divulgação dos dados científicos, posteriormente reafirmado em 2009 [8]. Os esforços da comunidade do genoma inspiraram o surgimento de uma "mesa redonda" de discussões sobre o tema da reprodutibilidade de experimentos científicos da área de ciência da computação [9]. Esse grupo faz recomendações para pesquisadores, agências de financiamento e editores de periódicos. Por exemplo, os pesquisadores são incentivados a gerar um identificador único para versões do *software* utilizado em um experimento, nos moldes do que é utilizado em sistemas de controle de versão como *git* e *svn* e a publicar essas e outras informações de forma *online* e permanente como em páginas da Internet pertencentes a uma universidade ou a um portal de compartilhamento de código como *SourceForge* [10] ou *GitHub* [11].

Tipicamente, as publicações da área de ciência da computação não apresentam as informações necessárias à reprodução de um experimento, como o conjunto exato de dados de entrada, as configurações de ambiente e procedimentos de inicialização e finalização da execução. Algumas causas para a omissão dessas informações são: falta de espaço no artigo, falta de disciplina dos escritores ou falta de interesse por parte dos leitores [6].

Em alguns casos, o problema é ainda maior, quando nem mesmo os próprios autores da pesquisa conseguem reproduzir os experimentos e obter os mesmos resultados. Em outros, mesmo que os devidos cuidados tenham sido tomados pelos autores para permitir a reprodução do experimento, alguns detalhes aparentemente irrelevantes, quando alterados, fazem com que os resultados sejam significantemente diferentes dos resultados originais. Mytkowicz e outros [12] mostram que a ordem com que os arquivos compilados são ligados produz um viés que pode favorecer o desempenho da execução de uma aplicação em um sistema A e prejudicá-lo em um sistema B. Ao comparar esses dois sistemas, podemos chegar à conclusão errônea de que o sistema A é melhor do que o B, quando uma simples alteração na ordem de ligação, aparentemente inofensiva, poderia fazer com que os resultados indicassem B como o melhor sistema.

O nosso laboratório desenvolve pesquisa nas áreas de compiladores e arquitetura de computadores. Os experimentos científicos típicos dessas áreas envolvem a compilação e execução de *softwares* (*benchmarks*) em diversas configurações de *hardware* e sistemas operacionais. A fim de evitar os problemas relacionados à validade e reprodução das descobertas científicas, desenvolvemos um banco de dados para armazenamento de informações sobre os experimentos e gerência dos dados experimentais. O banco de dados foi projetado de forma a ser genérico o suficiente para que sirva à maioria dos experimentos típicos desenvolvidos no laboratório, porém simples o suficiente para que seja amplamente adotado pelos pesquisadores.

As principais contribuições deste trabalho são:

- Um levantamento das informações necessárias à reprodução de experimentos científicos das áreas de compiladores e arquitetura de computadores.
- Um levantamento de casos de uso e de resultados experimentais tipicamente armazenados nesses experimentos.

978-0-7695-4847-0/12 $26.00 © 2012 IEEE
DOI

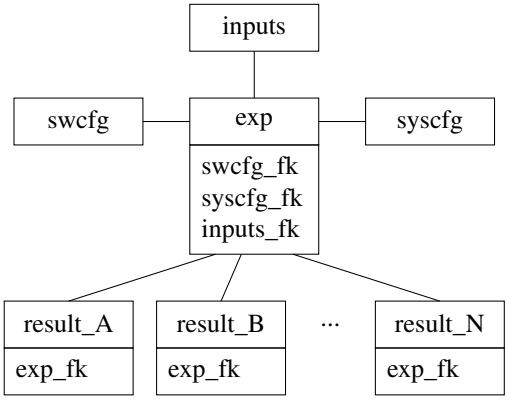

Figura 1. Tabelas centrais do banco de dados e chaves estrangeiras

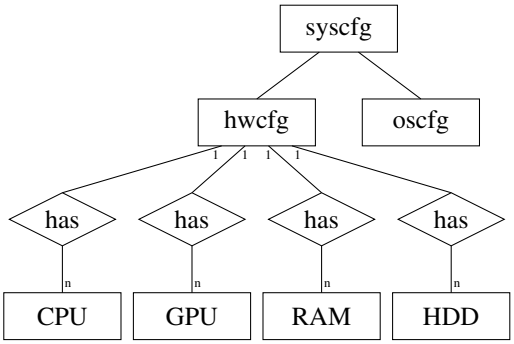

Figura 2. Tabelas de descrição do hardware

- A implementação de um banco de dados em linguagem SQL, disponível na Internet [13].

O texto está organizado da seguinte maneira. A Seção II apresenta uma visão geral sobre o projeto do banco de dados e as informações armazenadas para permitir a replicação dos resultados experimentais. A Seção III discute o armazenamento de resultados experimentais no banco de dados, A Seção IV discute casos de uso do banco de dados e, por fim, a Seção VI apresenta as conclusões e trabalhos futuros.

II. Projeto do banco de dados

O banco de dados é dividido em quatro grupos de entidades que armazenam informações sobre: a) o *software* e seu processo de construção; b) o *hardware* e o sistema operacional; c) os dados de entrada e d) os resultados. As relações de interdependência entre os grupos são armazenadas em uma entidade única central que representa um experimento científico. Dentre os grupos de entidades do banco de dados, o grupo dos resultados (d) é o único voltado para o compartilhamento de dados entre os pesquisadores, enquanto todos os demais são responsáveis pela garantia da reprodutibilidade dos experimentos. A Figura 1 ilustra as principais entidades do banco de dados e suas conexões. O uso das chaves estrangeiras obrigatórias `swcfg_fk`, `syscfg_fk` e `inputs_fk` garante que um experimento armazenado no banco de dados sempre guarda informações completas sobre a construção do *software*, sobre a configuração do sistema e sobre os dados de entrada, respectivamente. Por outro lado, a chave estrangeira `exp_fk`, nas entidades `result_*`, permite que múltiplas entradas de resultados estejam associadas a um único experimento, o que frequentemente é verdade. As subseções a seguir descrevem detalhadamente as entidades centrais e as demais tabelas que formam cada grupo.

A. Dados de configuração do sistema

As informações sobre o *hardware* e o sistema operacional da máquina onde um experimento foi executado são armazenadas em diversas tabelas organizadas ao redor da tabela `syscfg`. O preenchimento dessas tabelas exige a coleta de diversas informações sobre o sistema e embora esse seja um processo trabalhoso, as entradas são reaproveitadas por quaisquer experimentos que sejam realizados na mesma máquina e durante um ciclo de vida relativamente longo, uma vez que as configurações de *hardware* e sistema operacional raramente sofrem alterações.

A tabela `oscfg` armazena informações a respeito do sistema operacional da máquina onde um experimento é executado. Os campos `name`, `version` e `kernel` permitem a distinção entre diferentes sistemas. Informações adicionais que não tenham sido previstas durante o projeto do banco de dados podem ser armazenadas nos campos `build` e `comments`.

As informações sobre o *hardware* de uma máquina são armazenadas em diversas tabelas específicas organizadas ao redor da tabela `hwcfg`, que possui alguns campos para identificação única e descrição da placa-mãe. As informações sobre processadores de propósito geral (CPU) e gráfico (GPU), memória RAM e disco são armazenadas nas tabelas específicas. Tendo em vista que uma máquina pode possuir diversas instâncias de cada um desses componentes, a relação entre essas tabelas e a entidade `hwcfg` é do tipo um-para-muitos na modelagem do banco de dados. Portanto, a implementação precisa de um nível adicional de indireção, como pode ser visto na Figura 2.

Note que com essa organização, é possível adicionar uma quantidade infinita de componentes de *hardware* a uma configuração de sistema. Por exemplo, podemos ter uma máquina com 4 pentes de memória idênticos, duas CPUs idênticas, um disco rígido e duas GPUs distintas.

Diferentemente do restante do banco de dados, as tabelas `cpu`, `gpu`, `ram` e `hd` não possuem uma chave primária simples. A chave primária é a composição dos dois campos `manufacturer` e `model`. Essa decisão foi tomada para forçar o preenchimento dos dois campos de maneira correta, de forma que sempre seja possível utilizar o nome do fabricante e o modelo do equipamento para encontrar os *datasheets* oficiais dos componentes. Os demais campos nessas entidades são de preenchimento opcional, uma vez que sempre é possível obter as informações a partir dos *datasheets*. No entanto, o preenchimento dos campos adicionais é incentivado, pois facilita consultas subsequentes. Por exemplo, um usuário do banco de dados pode estar interessado nos resultados de quaisquer experimentos executados em computadores em

id	chave primária
name	nome do *software*
vendor	distribuidor
version	versão de distribuição
revision	*commit* no repositório
repository	URL do repositório
is_binary	*flag*

Tabela I
CAMPOS DA TABELA SWSOURCE

que a velocidade do barramento de memória é de 800MHz. Caso as informações adicionais da tabela `ram` não tenham sido preenchidas, essa consulta torna-se impossível sem o prévio conhecimento das características de todos os modelos de memória presentes no banco de dados.

B. Construção do software

Tipicamente, um experimento da área de sistemas de computação envolve a execução de um conjunto de *softwares* em uma ou mais configurações de *hardware*. Frequentemente, o *software* deve ser compilado durante o experimento e para que o processo de compilação possa ser replicado, garantindo a reprodutibilidade do experimento como um todo, a versão do código fonte, bem como o conjunto de ferramentas e regras de compilação precisam ser documentados. Na modelagem do banco de dados proposto, as informações relacionadas à construção do *software* são armazenadas nas tabelas `toolset` e `swsource` e conectadas através da tabela `swcfg`.

A tabela `swsource` armazena informações a respeito da origem de um *software*, seja essa origem código fonte ou um binário pré-compilado. A distinção entre esses dois tipos é feita através da *flag* `is_binary`. Há diversos campos para permitir a distinção entre diferentes *softwares*. O usuário do banco de dados deve prover a maior quantidade de informações possível, embora nem sempre todas as informações previstas pelo banco de dados estejam disponíveis. Por exemplo, é possível que um *software* possua um número de versão ou *release*, mas não possua informações sobre número da revisão no sistema de controle de versão (VCS) dos mantenedores. A Tabela I apresenta os campos componentes da tabela `swsource`. Os campos `id` e `name` são os únicos obrigatórios.

Um *software* pode precisar de diversas ferramentas para a sua construção. Esse comportamento se mapeia em uma relação do tipo um-para-muitos na modelagem do banco de dados e portanto precisa de uma implementação com um nível adicional de indireção, como pode ser visto na Figura 3. A entidade `tool` armazena ferramentas, enquanto o relacionamento `has` (mapeado para a tabela: `toolset_has_tool`) mapeia quais ferramentas pertencem a um determinado conjunto de ferramentas. Por sua vez, um único conjunto de ferramentas está associado à tabela `swcfg`. Podemos citar um exemplo comum de processos de compilação para facilitar o entendimento da necessidade de documentação do conjunto de ferramentas de construção. Os usuários da suíte de *benchmarks* SPEC CPU2006 [14] são forçados a preencher informações sobre

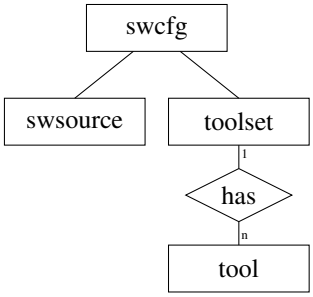

Figura 3. Tabelas de descrição do software

o compilador utilizado, porém não precisam se preocupar com a documentação de outras ferramentas utilizadas na compilação, como o montador e o ligador, possivelmente fornecidas pelo pacote binutils [15]. Mytkowicz e outros [12] mostram que o alinhamento de código, que é determinado por essas ferramentas, pode causar conflitos em mecanismos de *hardware*, como *cache* e *branch predictor*, possivelmente levando ao aparecimento de enviesamento nas medidas.

A entidade central `swcfg` concentra as informações sobre a construção de um *software* através de referências por chaves estrangeiras para as tabelas `swsource` e `toolset`. No entanto essas informações não são suficientes para permitir a reprodução exata do experimento, uma vez que os processos de compilação permitem, frequentemente, a passagem de parâmetros como nível de otimização ou habilitação/desabilitação de certos subcomponentes de *software*. Para armazenar essas informações, o usuário pode utilizar os campos `flags` e `options`, presentes na tabela `swcfg`.

C. Dados de entrada

A tabela `inputs` centraliza as informações sobre os dados de entrada, que são classificados em duas categorias: argumentos de execução e estímulos. Na maioria dos casos, a quantidade de argumentos e seu tamanho são pequenos o suficiente para serem armazenados diretamente no banco de dados. Essa tarefa é executada pela tabela `args`. Por outro lado, os estímulos de entrada geralmente são mais extensos, o que nos levou a optar por armazená-los em repositórios externos de dados. A referência à localização dos estímulos é feita através do caminho (URL) e da revisão no repositório externo, exceto nos casos em que os estímulos de entrada possuem um sistema de controle de versão, como é o caso dos *benchmarks* do SPEC. Nesses casos os campos `vendor`, `name`, `version` e `set` podem ser utilizados para identificar o conjunto de estímulos de entrada utilizados no experimento. A entidade responsável por armazenar essas referências é a tabela `stimuli`.

A execução de um *software* pode precisar de diversos arquivos de estímulos de entrada. Esse comportamento se mapeia em uma relação do tipo um-para-muitos no projeto do banco de dados que reflete em uma implementação com um nível adicional de indireção, como pode ser visto na Figura 4. Como os argumentos passados para a

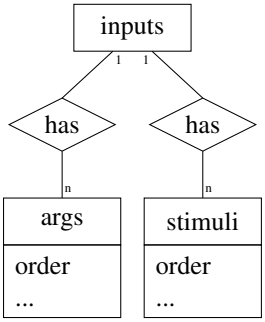

Figura 4. Tabelas de descrição das entradas

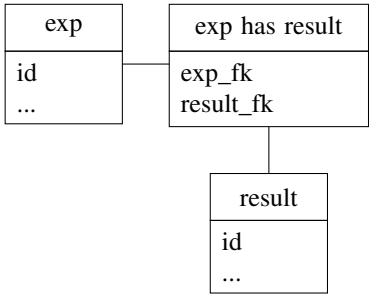

Figura 5. Tabelas de mapeamento inverso entre experimentos e resultados

aplicação podem variar a cada execução, as tabelas `args` e `stimuli` possuem um campo de enumeração para facilitar a associação mútua. Alternativamente, poderíamos projetar essas associações com o uso de chaves estrangeiras, porém essa abordagem não se mostrou adequada para os tipos de experimentos realizados no nosso laboratório, uma vez que a ordem das execuções e, consequentemente, a ordem dos argumentos e arquivos de estímulos precisa ser bem determinada.

Por fim, a entidade `inputs` centraliza as informações, provê o campo extra `comments` para anotação de instruções adicionais ou casos de uso não previstos e fornece um identificador único para referência na tabela central de informações sobre o experimento.

D. Juntando as partes

As informações contidas nos três grupos de entidades descritos nas seções anteriores são conectados à entidade central `exp`, que representa um experimento, de forma direta através de campos para armazenamento da chave primária desses grupos. Já a associação com os resultados experimentais é feita de maneira indireta, o que permite o preenchimento de diversas tabelas de resultados por um único experimento, como descrito na próxima seção.

III. Expansão do banco de dados

Embora as informações necessárias à reprodutibilidade sejam comuns a diversos experimentos científicos da área de compiladores e arquitetura de computadores, os dados coletados e análises frequentemente não o são. Como um dos principais objetivos deste trabalho é promover o reuso das informações necessárias à reprodutibilidade dos experimentos, os usuários tem liberdade para criar as próprias tabelas de resultados e associá-las a uma entrada na tabela `exp`, garantindo a reprodutibilidade, sem perdas na flexibilidade de uso.

A associação entre experimento e resultados é feita de maneira indireta, isto é, uma entrada na tabela `exp` não guarda referências para as tabelas de resultados que preenche, mas as tabelas de resultados devem manter uma chave estrangeira referente ao experimento que gerou o resultado. A Figura 1 ilustra a situação.

Essa organização do banco de dados permite que, dada uma tabela de resultados, possamos encontrar os experimentos que os geraram, tornando possível a reprodução do experimento e a verificação da validade dos dados. No entanto, é possível que um usuário deseje fazer buscas de maneira inversa, isto é, o usuário pode selecionar um determinado experimento e estar interessado nos resultados gerados por ele.

Tendo apenas as tabelas de resultados e a tabela `exp`, uma busca pelos resultados gerados por um experimento exigiria a consulta do campo `exp_fk` em todas as tabelas de resultados no banco de dados, mesmo aquelas que nunca são preenchidas pelo experimento em questão. Para evitar a sobrecarga associada à quantidade de consultas que essa operação geraria, o banco de dados conta com duas tabelas para fazer o mapeamento inverso, como pode ser observado na Figura 5.

A tabela `result` guarda os nomes das tabelas de resultados presentes no banco de dados, bem como uma descrição dos dados contidos nela, enquanto a tabela `exp_has_result` faz o mapeamento, através de chaves estrangeiras, para a tabela `exp`. Com essa abordagem, um experimento mantém a habilidade de preencher múltiplas tabelas de resultados, basta para isso, que o usuário insira múltiplas entradas na tabela de mapeamentos (`exp_has_result`).

IV. Casos de uso

Apresentamos aqui os casos de uso que ajudaram a definir a estrutura interna do banco de dados proposto. Esses são alguns dos casos típicos de experimentos no Laboratório de Sistemas de Computação do Instituto de Computação da Unicamp.

A. SPEC

SPEC CPU2006 é uma suíte de *benchmarks* voltada para análise de processadores e compiladores. Os *benchmarks* contidos na suíte são aplicações com cargas de trabalho intensivas em processamento [14]. A distribuição dessas aplicações é feita sob a forma de código fonte, o que permite a experimentação de compiladores e outros programas e bibliotecas relacionados à *toolchain* de construção de *software*. A suíte SPEC CPU também é amplamente utilizada para avaliação de arquiteturas de computadores. No SPEC, a execução dos *benchmarks* é feita através de *scripts* automatizados que compilam o código a partir do fonte e executam o binário gerado utilizando arquivos de estímulos de entrada predefinidos.

Um dos projetos comumente realizados no nosso laboratório é a análise dos efeitos causados pelas *flags* de compilação no desempenho da execução de aplicações. Compiladores como o GCC e o LLVM implementam diversos passos de otimização da compilação que podem ser selecionados como grupos pré-estabelecidos (O1, O2, O3, etc.) ou individualmente. O problema de decisão da ordem e número de repetições com que os passos de otimização devem ser aplicados a fim de produzir ganhos ótimos de desempenho para quaisquer aplicações é difícil, se não impossível, de resolver [16]. Apesar de existirem ordens que, na maioria dos casos, apresentam bons resultados (como os grupos pré-estabelecidos do GCC e do LLVM), alterações nessa ordem e a remoção ou inserção de passos podem ser favoráveis para aplicações ou sistemas específicos. Portanto o banco de dados deve oferecer um campo (`flags`) para preenchimento dessa informação.

Além da seleção dos passos de otimização utilizados na compilação, a versão dos algoritmos que os implementam também reflete na qualidade do programa binário gerado. Essa é uma informação fácil de ser obtida, uma vez que os compiladores tipicamente possuem um número de versão, que pode ser armazenado no campo `version` da tabela `tool`, junto com as outras informações que o identificam, `name` e `vendor`. A versão de outras ferramentas utilizadas na compilação também deve ser armazenada na tabela `tool`: o ligador, que pode aplicar otimizações de tempo de ligação, e a biblioteca padrão C, que implementa funções básicas para a linguagem C, são exemplos de ferramentas que podem afetar o desempenho de uma aplicação.

O preenchimento das tabelas que indicam a origem do *software* (`swsource`) e os dados de entrada (`stimuli`) é relativamente simples no caso das suítes de *benchmarks* SPEC, pois o SPEC define essas informações de maneira oficial. Suponha que no nosso caso de uso, tenhamos usado a versão CPU2006v1.2 do SPEC, nesse caso devemos preencher os campos `vendor` e `version` das duas tabelas, com os valores "SPEC"e "CPU2006v1.2", respectivamente. Adicionalmente, devemos preencher o campo `set` da tabela `stimuli` com uma indicação do conjunto de dados utilizados no experimento, uma vez que o SPEC fornece três opções de dados de entrada: `ref`, `train` e `test`.

B. ArchC

ArchC é uma linguagem de descrição de arquitetura (ADL) baseada em SystemC [17], que facilita o desenvolvimento de processadores e conjuntos de instrução (ISA), de hierarquias de memória e de outros aspectos relacionados a arquitetura de computadores [18]. No fluxo completo de desenvolvimento de uma nova arquitetura utilizando o ArchC, o usuário começa com a descrição do conjunto de instruções (AC_ISA) e dos recursos do sistema (AC_ARCH). Essas descrições são feitas na linguagem ArchC e armazenadas em dois arquivos distintos. Em seguida o usuário utiliza as ferramentas do ArchC para gerar, a partir das duas descrições, um simulador (interpretado ou compilado) da nova arquitetura e um montador. Por fim, o usuário utiliza o montador para gerar código binário e utilizá-lo no simulador para testar a arquitetura descrita.

Do ponto de vista de um usuário do ArchC, não de um desenvolvedor, o uso do ArchC é semelhante à execução de *benchmarks* da suíte SPEC (Seção IV-A), pois envolve a compilação de *software* (o modelo em ArchC) e a execução do programa gerado (o simulador). No entanto, temos nesse caso uma etapa adicional de montagem, prévia à simulação.

A etapa de compilação é representada no banco através das tabelas `swsource` e `tool`. A ferramenta ArchC integra o conjunto das ferramentas de compilação (junto com gcc, make, bison, flex, binutils e systemc), e deve ser inserida na tabela `tool` com sua informação de versão. Já os modelos (AC_ISA e AC_ARCH) compõem o *software* a ser compilado e são adicionados à tabela `swsource`. Como os modelos podem estar em fase de desenvolvimento, é provável que ainda não possuam informações de *release* como os *benchmarks* do SPEC. Nesse caso, o usuário é aconselhado a utilizar os campos `revision` e `repo` para preencher informações sobre o sistema de controle de versão onde os modelos são armazenados.

Uma vez que o simulador executa código no formato binário, podemos ter uma etapa adicional de compilação de *software*, caso o programa a ser executado pelo simulador esteja na forma de código fonte. Para facilitar o entendimento do caso de uso, vamos assumir que o programa já se encontra previamente compilado e ligado, de forma que precisamos apenas utilizar o montador gerado pela compilação do modelo ArchC. Nesse caso, o mapeamento para o banco de dados das informações sobre o programa a ser simulado é feito através das tabelas em torno da tabela `inputs`. A tabela `stimuli` guarda informações sobre o programa pré-compilado que precisa ser montado e sobre os arquivos de estímulo utilizados na simulação. Já a tabela `args` armazena os argumentos que são passados para o montador e o simulador.

C. SPEC sobre o DynamoRIO

DynamoRIO é um tradutor dinâmico de binários que facilita a instrumentação, tradução e otimização de código através da disponibilização de uma interface de programação que abstrai os detalhes internos de sua implementação [19], [20]. Máquinas virtuais, grupo de aplicações do qual tradutores dinâmicos de binários fazem parte, são frequentemente utilizadas em projetos de pesquisa da área de arquitetura de computadores, pois, dentre outros motivos, emulam máquinas reais, permitindo a execução de testes logo nas primeiras etapas do desenvolvimento de novas arquiteturas de computadores. Os simuladores gerados pelo ArchC, descrito na Seção IV-B, são outros exemplos de máquinas virtuais.

Dada a importância dos tradutores dinâmicos de binários para os projetos de pesquisa em arquitetura de computadores, os algoritmos de tradução são objeto de estudo

do nosso laboratório e para avaliar a eficiência desses algoritmos, utilizamos os *benchmarks* da suíte SPEC CPU como estímulos de entrada para testes do DynamoRIO. Diferentemente do apresentado na Seção IV-A, nesse caso de uso, os *benchmarks* do SPEC não são utilizados como código a ser compilado e executado, mas como estímulos de entrada para a execução de outro *software*, o DynamoRIO.

No banco de dados, o DynamoRIO é tratado como o *software* que deve ser compilado e executado, portanto, preenche as tabelas do grupo organizado ao redor da tabela `swcfg`. Já os *benchmarks* do SPEC são tratados como dados de entrada e preenchem as tabelas ao redor da tabela `inputs`. Embora o projeto de pesquisa inclua a execução de todos os *benchmarks* do SPEC sobre o DynamoRIO, isto é, como estímulos de entrada, o método mais simples de organizar a informação no banco de dados é criar diversos experimentos menores, onde apenas um dos *benchmarks* é utilizado. Por fim, como a execução do DynamoRIO não é automatizada por *scripts* como no caso do SPEC, devemos preencher a tabela `args` com informações sobre os argumentos utilizados para execução do experimento.

V. Trabalhos relacionados

A ideia de documentar experimentos científicos com detalhes suficientes para garantir sua reprodutibilidade não é nova. Em trabalhos colaborativos, como o projeto genoma [8], essa documentação é necessária para garantir a confiança nos dados obtidos pelos demais pesquisadores envolvidos e para permitir a verificação dos resultados. Em uma escala reduzida, os dados produzidos no nosso laboratório podem ser reaproveitados internamente, ou mesmo compostos, para gerar novos resultados.

Inspirados pelos esforços da comunidade do projeto genoma, porém voltada para a área da ciência da computação, a "mesa redonda" de discussões sobre compartilhamento de código e dados [9] apresenta uma série de recomendações para melhorar o nível de reprodutibilidade das pesquisas desenvolvidas. O banco de dados apresentado neste trabalho é uma ferramenta que facilita o cumprimento dessas recomendações, pois permite a identificação dos códigos utilizados para confecção dos experimentos, bem como instruções para construção e execução das aplicações.

Uma outra categoria de ferramentas para controle de experimentos é o uso de *workflows*. O projeto WASA [21] é um exemplo de ferramenta baseada em *workflows* que integra técnicas e ferramentas de gerenciamento de experimentos, de dados e de aplicações. O sistema permite a documentação de experimentos de forma genérica e atende a pesquisadores de diversas áreas. Nosso sistema difere do WASA e de outras ferramentas baseadas em *workflows* por ser específico para a área de compiladores e de arquitetura de computadores e por utilizar o banco de dados como a ferramenta central. Essa abordagem simplifica o projeto do banco de dados, o que facilita a sua adoção por novos usuários, visto que a quantidade de informação necessária para utilizar o mesmo é reduzida. *Scripts* para automação da inserção e consulta aos dados podem ser utilizados e são encorajados, porém não fazem parte do sistema de gerenciamento dos dados.

VI. Conclusão e trabalhos futuros

Tipicamente, as publicações da área de ciência da computação não apresentam as informações necessárias à reprodução de um experimento, como o conjunto exato de dados de entrada, as configurações de ambiente e procedimentos de inicialização e finalização da execução [6].

Através do estudo de casos de confecção e execução de experimentos típicos do nosso laboratório, conseguimos levantar um conjunto de informações necessárias para garantir a reprodutibilidade dos resultados experimentais. A partir desses estudos, desenvolvemos um banco de dados para armazenar estas informações bem como os resultados dos experimentos e facilitar a gerência dos dados científicos.

O projeto do banco de dados se mostrou genérico o suficiente para servir à maioria dos experimentos, ainda assim simples de forma a facilitar a sua adoção pelos membros do laboratório e permitir uma maior colaboração e compartilhamento de dados e resultados entre os pesquisadores.

Convidamos a comunidade de pesquisadores na área de sistemas de computação a utilizar o banco de dados desenvolvido e propor melhorias para fomentar o compartilhamento e a reprodução de resultados experimentais.

VII. Agradecimentos

Agradecemos à Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) pelo apoio financeiro recebido para realizar este trabalho - processos número 2011/16468-6 e 2011/00901-2.

Referências

[1] "American Physics Society," http://www.aps.org.

[2] S. Fomel and J. F. Claerbout, "Reproducible Research," *Computing in Science & Engineering*, 2009.

[3] K. Chang, "Nobel Laureate Retracts Two Papers Unrelated to Her Prize," *The New York Times*, 2010, http://www.nytimes.com/2010/09/24/science/24retraction.html.

[4] C. Zimmer, "It's Science, but Not Necessarily Right," *The New York Times*, 2011, http://www.nytimes.com/2011/06/26/opinion/sunday/26ideas.html.

[5] N. Wade, "It May Look Authentic; Here's How to Tell It Isn't," *The New York Times*, 2006, http://www.nytimes.com/2006/01/24/science/24frau.html.

[6] P. Vandewalle, J. Kovačević, and M. Vetterli, "Reproducible Research in Signal Processing [What, why, and how]," *IEEE Signal Processing Magazine*, 2009.

[7] J. Freire, P. Bonnet, and D. Shasha, "Computational Reproducibility: State-of-the-Art, Challenges, and Database Research Opportunities," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2012.

[8] Toronto International Data Release Workshop Authors, "Prepublication data sharing," *Nature*, 2009.

[9] Yale Law School Round Table on Data and Code Sharing, "Reproducible Research," *Computing in Science & Engineering*, 2010.

[10] "SourceForge," http://sourceforge.net.

[11] "Github. Social Coding," https://github.com.

[12] T. Mytkowicz, A. Diwan, M. Hauswirth, and P. F. Sweeney, "Producing Wrong Data Without Doing Anything Obviously Wrong!" in *Proceedings of the International Conference on architectural support for programming languages and operating systems*, 2009.

[13] "Scripts SQL de criação do banco de dados proposto," http://user.lsc.ic.unicamp.br/~gabriel/published/sharedexp.

[14] "SPEC: Standard Performance Evaluation Corporation," http://www.spec.org.

[15] "GNU Binutils," http://www.gnu.org/software/binutils.

[16] S. S. Muchnick, *Advanced Compiler Design and Implementation*. Morgan Kaufmann Publishers, 1997.

[17] J. Bhasker, *A SystemC Primer*. Star Galaxy Publishing, 2004.

[18] S. Rigo, G. Araujo, M. Bartholomeu, and R. Azevedo, "ArchC: a systemC-based architecture description language," in *Proceedings of the Symposium on Computer Architecture and High Performance Computing*, 2004.

[19] D. Bruening, T. Garnett, and S. Amarasinghe, "An infrastructure for adaptive dynamic optimization," in *Proceedings of the International Symposium on Code Generation and Optimization*, 2003.

[20] D. Bruening, "Efficient, transparent, and comprehensive runtime code manipulation," Ph.D. dissertation, Massachusetts Institute of Technology, 2004.

[21] C. Medeiros, G. Vossen, and M. Weske, "WASA: A Workflow-Based Architecture to Support Scientific Database Applications," in *Database and Expert Systems Applications*. Springer Berlin / Heidelberg, 1995.