ArchC Platform Manager
Um gerenciador de pacotes para plataformas de sistemas embarcados

Relatório Final de Atividades

PIBIC/CNPq

Instituto de computação - UNICAMP

Campinas, 8 de agosto de 2013

Orientador: Rodolfo Jardim de Azevedo

Aluno: Matheus Ferreira Tavares Boy

Sumário

1 Resumo

A atividade de Iniciação Científica desenvolvida pelo aluno neste semestre foi uma continuação da atividade do semestre passado. O trabalho no *ACPM* foi continuado, assim como as atividades relacionadas ao *MPSoCBench* (antigo *ESLBench*). Vale ressaltar que o *MPSoCBench* é um *benchmark* que visa medir o desempenho de simuladores de plataformas de sistemas embarcados. O *MPSoCBench* foi uma atividade desenvolvida em conjunto com a aluna de doutorado Liana Duenha, do LSC (Laboratório de Sistemas de Computação) do IC (Instituto de Computação).

A primeira parte foi feita através de melhorias e adaptações no código do *ARP*, além da implementação de novas funcionalidades. Foi feita uma análise de pontos de interesse e melhoria no código original do *ARP*, e esses foram tratados e corrigidos. Além disso, foram estudadas e implementadas novas funcionalidades para o gerenciador. Uma versão estável do *ACPM* já se encontra no site do *ArchC* (archc.org) para download.

A segunda tarefa consistiu na criação de um script para atuar de *frontend* para o *MPSoCBench*. O script foi criado do zero, visto que não existia um código anterior no qual se basear. Essencialmente, o script permite uma interface mais amigável para quem for rodar o *benchmark*.

2 Introdução

O projeto visa atualizar e melhorar a atual ferramenta de gerenciamento de pacotes do *ArchC*[2][9], o *ARP* (*ArchC* Reference Platform)[1], transformando-o em *ACPM* (*ArchC* Platform Manager).

Para isso, uma série de providências e melhorias foram tomadas de modo a realizar essa transição de *ARP* para *ACPM*. Além disso, também foi desenvolvido um trabalho em paralelo com o objetivo de desenvolver um *benchmark*. Estabelecendo uma visão geral da atividade, foi desenvolvido um trabalho em duas frentes, e simultaneamente.

Os principais objetivos do projeto consistem em:

1. Incluir suporte a múltiplos repositórios remotos, de modo a favorecer a obtenção de componentes;
2. Melhorar o método de empacotamento dos componentes, para facilitar a obtenção e busca dos mesmos;
3. Possibilitar a obtenção e instalação de ferramentas como *SystemC*[8], *ArchC*, cross-compilers através do *ACPM*, facilitando o trabalho dos usuários do *ArchC*;
4. Criar uma interface gráfica para o *ACPM* através de um browser, funcionando como uma interface web local;
5. Suporte no desenvolvimento e distribuição do *MPSoCBench*.

Os objetivos concluídos foram o 1, 2 e 5. Os objetivos 3 e 4 estavam em andamento até o cancelamento da bolsa em maio deste ano, não tendo sido concluídos por conta disso. Os objetivos 1 e 2 expandiram as funcionalidades

do *ACPM* de modo a atingir uma versão estável, já disponível para download no site do *ArchC*. O objetivo 5 tratava das questões ligadas ao *MPSoCBench*, e foi concluído no sentido de que o *benchmark* encontra-se numa versão estável e já disponível para download no site do *ArchC*. Desse modo, pode-se dizer que os objetivos cruciais da atividade de iniciação científica foram concluídos com sucesso. As atividades que ficaram pendentes tinham um caráter acessório, de modo que sua incompletude não afeta o funcionamento básico do *ACPM*.

Houve esse direcionamento das atividades de modo a cumprir o cronograma proposto no projeto da atividade de Iniciação Científica enviado ao CNPq.

Este relatório está estruturado em seções. A seção de "Material e métodos" explica em detalhes as atividades desenvolvidas de modo a atingir os objetivos propostos. A seção de "Resultados" esclarece o que se atingiu com o desenvolvimento das atividades propostas, e a seção de "Discussão e conclusões" faz uma avaliação geral de todas as atividades desenvolvidas até o momento.

3 Material e métodos

O *ACPM* é um gerenciador de pacotes para simuladores de plataformas de sistemas embarcados. Seu principal objetivo é obter pacotes num repositório remoto, via web, e desempacotá-los localmente, dentro de uma estrutura específica. Além disso, o *ACPM* permite gerar diversas plataformas diferentes com um conjunto de componentes limitado. Isso vem da estrutura organizacional do gerenciador, herdada da *ARP*, que pode ser vista na Figura 1.

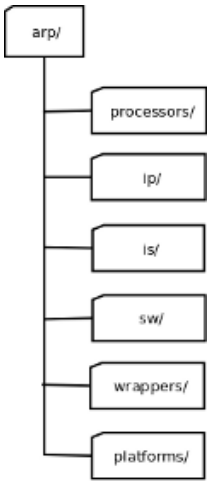

Figura 1: Estrutura de diretórios do ARP

Por exemplo, uma estrutura que possua como componentes um processador, uma memória e um barramento pode gerar diversas plataformas distintas com apenas esses 3 componentes. Na Figura 2 pode-se ver 2 plataformas distintas geradas com exatamente os mesmos componentes, sendo uma plataforma *single-core* e outra *multicore*. Para gerar essas plataformas, basta criar um arquivo para cada plataforma dentro do diretório *platforms* da estrutura do gerenciador, instanciando os componentes utilizados.

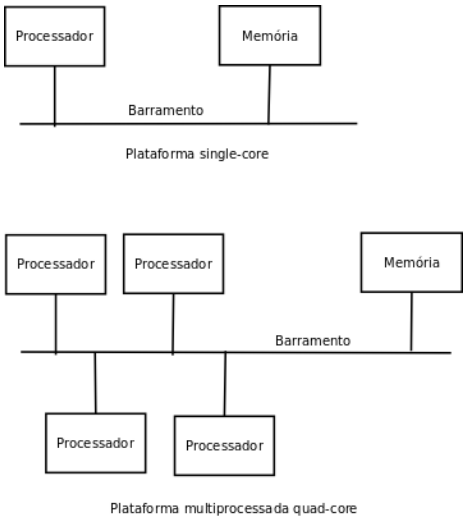

Figura 2: Exemplos de plataformas

O *ACPM* possui vários comandos para executar diversas tarefas diferentes. As funções disponíveis no gerenciador são:

- `create` - gera um *template* de componente
- `get` - obtém um componente do repositório
- `list` - lista os componentes disponíveis localmente
- `listrepo` - lista os componentes disponíveis no repositório
- `pack` - empacota um componente
- `put` - empacota um componente e o copia para um diretório
- `repo` - gera o arquivo de indexação do repositório
- `start` - gera a estrutura organizacional, arquivos de configuração e Makefiles
- `unpack` - desempacota um componente

Todas essas funções constavam no gerenciador original do *ARP*, mas todas foram alteradas de alguma forma. Além disso, algumas modificações gerais no código do gerenciador foram feitas. As alterações no código serão descritas a seguir.

Inicialmente, estudou-se o script original do *ARP*, que foi implementado pelo aluno Matheus Nagliati, em atividade de Estudo Dirigido orientada pelo professor Rodolfo no primeiro semestre de 2010. Após devida compreensão de código e domínio de uso do gerenciador original, iniciou-se um processo de detecção de pontos de interesse para melhoria ou modificação no código.

Vale mencionar que o gerenciador do *ARP* originalmente foi implementado em *Python* e foi escolhido manter essa como ferramenta de implementação do *ACPM*, por conta da familiaridade do aluno com essa linguagem, além da própria

praticidade do *Python* para implementações não muito grandes, como a do próprio *ARP* e *ACPM*. Outro ponto forte do *Python* é a grande quantidade de bibliotecas e módulos padrão da linguagem, facilitando muitas tarefas, como geração de arquivos zip, chamadas para o interpretador de comandos do sistema operacional, e acesso à web. Além disso, foi escolhido o *Git* como controle de versão para o projeto, sendo hospedado inicialmente num repositório no github.com e depois num repositório privado no LSC.

Após o estudo do código do gerenciador, foram detectados muitos pontos passíveis de melhoria no mesmo. Existia um tratamento muito superficial de exceções, tornando o código muito pouco tolerante à falhas. Muitas vezes eram emitidas verbosas exceções do interpretador do *Python*, que não davam pista alguma de qual o real problema que lançou a exceção. Um ponto interessante a ser mencionado é que, mesmo sendo um gerenciador de pacotes com repositório na web, quase nenhuma exceção decorrente de métodos que buscavam dados na web era tratada num bloco de tratamento de exceções. Além disso, nenhuma operação de leitura ou escrita de arquivo possuía tratamento de exceção.

Depois de detectados esses pontos de interesse, foi iniciado um minucioso trabalho de blindagem do código. Todos os pontos passíveis de exceções foram inseridos em blocos `try/except`, inclusive diferenciando-se alguns tipos específicos de exceções, de modo a receberem tratamento apropriado ao invés de um tratamento genérico e inespecífico. Contudo, incialmente, o tratamento de exceções se restringia apenas a emitir uma mensagem mais clara ao usuário do que apenas uma exceção do interpretador do *Python*. Posteriormente, algumas exceções receberam um tratamento mais apropriado, emitindo uma mensagem de aviso para o usuário, mas sem encerrar a execução do gerenciador. Após algumas iterações de testes e correções no código, a versão original do gerenciador foi declarada como blindada e apta a receber novas funcionalidades e outras melhorias no código.

Estando com o código original devidamente blindado e com uma certa tolerância a falhas, iniciou-se um processo de implementação de novas funcionalidades, além da expansão e melhoria de funcionalidades já presentes.

A primeira funcionalidade que foi proposta foi o suporte a múltiplos repositórios por parte do gerenciador. O suporte a múltiplos repositórios foi implementado de modo que sempre que o gerenciador fosse rodado ele selecionasse o primeiro repositório válido num arquivo de fila de prioridades de repositórios, facilmente editável pelo usuário. Contudo, na instalação do gerenciador, um repositório padrão, hardcoded, é utilizado para inicializar o arquivo de fila de prioridades de repositórios; depois disso o usuário pode editar o arquivo com outros repositórios de sua preferência. O funcionamento desse recurso é um tanto simples. Essencialmente, o gerenciador tenta acessar sequencialmente cada repositório listado no arquivo e seleciona o primeiro que emitir uma resposta HTTP 200 (OK). Se nenhum repositório retornar uma HTTP 200, o repositório hardcoded é selecionado.

Depois disso, foi proposta a mudança do formato de compactação dos pacotes de tar para zip. Isso se deu devido ao fato do gerenciador originalmente gerar os pacotes chamando tar(1) num subprocesso aberto pelo interpretador de comandos do sistema operacional, decisão de projeto que restringe o uso do gerenciador a um ambiente POSIX (ou *quasi*-POSIX). Como o *Python* suporta arquivos zip através de uma biblioteca padrão, a implementação foi relativamente simples. Foi necessária a criação de uma classe auxiliar para facilitar o

manuseio de tais arquivos, mas nada intrinsecamente complexo. Contudo, foi necessária uma mudança na instalação do gerenciador, já que era obtido um arquivo tar com toda a estrutura de diretórios do *ARP* inclusa e o formato zip não suporta explicitamente a compressão de uma árvore de diretórios. O que se faz é alterar o nome dos arquivos de modo a concatenar o path de diretórios do arquivo com o próprio nome do arquivo. Sendo assim, zip não suporta a compressão de diretórios vazios, não suportando portanto a inclusão da estrutura básica de diretórios do *ARP* no arquivo de inicialização/instalação do gerenciador. A solução encontrada foi criar dinamicamente os diretórios vazios da estrutura básica na função de instalação do gerenciador, através da biblioteca padrão do *Python*. Resolvidas essas questões, foi totalmente implementada a substituição do formato de pacotes de tar para zip.

Outra funcionalidade proposta foi a implementação de um parser de YAML no próprio código do gerenciador. YAML é um formato de serialização de codificação de dados facilmente legível por seres humanos. O gerenciador utiliza YAML para listar os componentes armazenados num repositório. Originalmente, o parsing dos dados serializados com YAML era feito utilizando-se uma biblioteca desenvolvida pelo próprio projeto YAML, contudo, não padrão do Python. Sendo assim, para o gerenciador funcionar de forma apropriada, era necessária a instalação dessa biblioteca, algo não-trivial num ambiente não-POSIX. Desse modo, foi implementado um parser de YAML, tarefa não muito complexa, mas também não tão trivial. Enquanto as funcionalidades anteriores tinham como maior desafio um conhecimento da biblioteca padrão do *Python*, a implementação do parser trouxe desafios num nível algorítmico, resultando em problemas mais interessantes para resolver. O parser foi implementado com sucesso e foi um dos problemas mais interessantes encontrados e resolvidos no *ACPM*.

Posteriormente, foi feita uma alteração na funcionalidade de empacotamento de componentes `pack`, na qual foi incluída a opção de se empacotar todos os componentes, separadamente, com apenas um único comando. Isso foi implementado utilizando-se chamadas recursivas na função empacotadora à medida que se caminha a estrutura padrão do *ACPM* (`platforms`, `processors`, `ip`, `is`, `sw` e `wrappers`), passando como argumento o tipo do componente e o próprio componente encontrado. Manteve-se a possibilidade de gerar um único arquivo contendo todos os componentes.

Implementou-se uma forma inicial do `put` para o *ACPM*, visto que essa funcionalidade não havia sido implementada no código original. O conceito do *put* seria permitir ao usuário do *ACPM* que submetesse seus próprios componentes a um repositório. A forma inicial implementada não permite, explicitamente, a submissão de componentes a um repositório, contudo, o usuário pode fazer com que essa funcionalidade se comporte assim. Essencialmente, o *put* foi implementado como uma chamada de *pack* para um determinado componente e com a subsequente cópia do componente empacotado para um determinado diretório. O diretório-alvo do `put` pode ser alterado através de um arquivo de configuração gerado automaticamente pelo *ACPM*, mas de fácil edição, de modo que o usuário possa especificar um local mais apropriado para a submissão de seus componentes. Desse modo, se o usuário alterar o arquivo de configuração para apontar um diretório que seja de fato um repositório, o `put` funciona como se esperaria no conceito inicial.

Além dessas funcionalidades novas, foram feitas algumas correções no código

que vão além de blindar o código e torná-lo independente de plataforma. Algumas funções do código original do gerenciador redefiniam funções builtin do *Python*, como `list()`. Os nomes dessas funções foram trocados de modo que nada da linguagem fosse redefinido.

Depois de feitas todas essas alterações e correções no código do *ACPM*, o mesmo foi declarado como estando em sua primeira versão estável e disponibilizado para download no site do *ArchC*.

De certa forma, outro projeto foi tocado em paralelo ao *ACPM* na atividade de Iniciação Científica. Trata-se do *MPSoCBench*, *benchmark* proposto para avaliar o desempenho do escalonador do *SystemC*, trabalho conjunto do aluno com a aluna de doutorado Liana Duenha. Essencialmente, a aluna implementou a maior parte do *benchmark*, descrevendo as plataformas para teste em *ArchC* e *SystemC*, e rodando softwares de alguns *benchmark*s populares, como ParMiBench[6], SPLASH-2[10] e PARSEC[3]. O aluno trabalhou no script de *frontend* do *benchmark*, utilizando técnicas empregadas no *ACPM*, e outras técnicas desenvolvidas especialmente para o *benchmark*. Apesar da carga de trabalho no *frontend* ter sido relativamente menor que no *backend*, surgiram diversos problemas interessantes para resolver ao longo da implementação do script. Alguns problemas interessantes tratados foram: geração dinâmica de Makefile, esquemas de paralelização de execução das plataformas e *parsing* de argumentos para o script. Além do trabalho de implementação do script de *frontend*, foi feito um estudo teórico através da leitura de vários papers relacionados a *benchmark*s existentes e populares, dentre eles o MiBench[5], ParMiBench, PARSEC, SPLASH-2, MediaBench[7] e MEVBench[4]. Em suma, a tarefa de implementação do *frontend* para o *benchmark* foi bem interessante e trouxe alguns conceitos que serão aplicados ao *ACPM*.

Inicialmente, um breve panorama geral do *MPSoCBench*. O *benchmark* é estruturado de modo a ter diversas implementações de processadores descritos em *ArchC*: ARM, Mips, PowerPC e Sparc. Além disso, existem diversas plataformas, tanto single-core quanto multicore, além de plataformas heterogêneas, contendo no mínimo um processador de cada tipo na mesma plataforma. Vale ressaltar que as plataformas podem ter de 1 até 64 processadores, e pode ser escolhido o tipo de interconexão da plataforma entre *NoC* (network on chip) e *router*, além do modo de timing do dispositivo de interconexão: *loosely timed* ou *appromately timed*. Foram reunidos diversos softwares para serem executados nas plataformas, todos oriundos de *benchmark*s populares consagrados no meio acadêmico. O objetivo principal do script de *frontend* é automatizar e agilizar a geração e execução dos simuladores de plataformas, facilitando a execução do *benchmark* por terceiros.

O primeiro problema a ser abordado foi a geração dinâmica de Makefile. Antes, o Makefile tinha que ser editado manualmente e era necessária a passagem manual de regras de make(1) para a linha de comando. Isso tornava o processo de geração e execução das plataformas muito lento e maçante. Sendo assim, foi pensada uma forma de gerar dinamicamente Makefiles e de executar as devidas regras de make(1) através do script. Como as plataformas e softwares são pré-definidas, é relativamente fácil gerar um arquivo de Makefile automaticamente. Contudo, por simplicidade, resolveu-se dividir o Makefile original em 3: Makefile (gerado dinamicamente), Makefile.rules (com todas as regras de compilação e execução de plataformas) e Makefile.conf (com os paths para *SystemC*, *ArchC*, TLM e cross-compilers). Desse modo, o Makefile dinâmico

dá um `include` no Makefile.rules e no Makefile.conf. Inicialmente, o script foi escrito de forma a receber argumentos que especificavam qual processador, número de cores, software e dispositivo de interconexão a comporem a plataforma, além de aceitar a geração sequencial de todas as possibilidades de processador, número de cores, software e interconexão de modo a automatizar ainda mais a geração e execução das plataformas. Posteriormente, a ideia de passagem de argumentos foi amadurecida para possibilitar que as plataformas sejam geradas sem necessariamente executá-las. Isso se relaciona com a solução de paralelização da execução de plataformas (que será explicada mais adiante). Foi incluído um argumento para permitir que se meça o consumo de energia nas plataformas com processadores MIPS e SPARC por meio da biblioteca PowerSC e esse argumento resulta numa alteração no Makefile gerado dinamicamente, emitindo uma variável de ambiente adicional.

Solucionada a questão da geração dinâmica de Makefile, o próximo problema a ser abordado foi a paralelização da execução das plataformas. Inicialmente, executava-se uma plataforma por vez, e no próprio diretório raiz do *MPSoC-Bench*. Isso gerava problemas ao se tentar executar mais de uma plataforma simultaneamente, já que o Makefile era sobrescrito e variáveis de ambiente eram modificadas. A solução encontrada foi criar um diretório adicional na estrutura básica do *ARP*, chamado de `rundir` e dentro desse diretório existiria um subdiretório para cada plataforma gerada, onde elas serão executadas. Desse modo, a execução pôde ser paralelizada. A alteração feita no script foi a criação do `rundir` associado à geração da plataforma. Isso, contudo, não permite a paralelização da geração da plataforma, já que a compilação é feita no próprio diretório do *MPSoCBench*. Entretanto, a paralelização da geração de plataformas não é algo tão crucial quanto a paralelização da execução, visto que se pode gerar todas as plataformas do benchmark com um único comando para o script.

Ainda surgiu uma outra questão no *MPSoCBench*. Para rodarmos o *benchmark* no *cluster* do LSC, nós precisávamos submetê-lo ao gerenciador de tarefas *condor*. O *condor* é baseado em arquivos de configuração das tarefas, que podem ser gerados dinamicamente através de um script com relativa facilidade. Desse modo, inclui-se no script do *MPSoCBench* um trecho de código para gerar o arquivo de configuração da tarefa para o *condor* dentro do `rundir` da plataforma.

As plataformas heterogêneas são um caso muito particular no *benchmark* e são extremamente dignas de nota. Uma plataforma heterogênea pode conter de 4 até 64 *cores*, sendo que possuem processadores de todos os tipos. Ou seja, uma plataforma com 4 *cores* possui um ARM, um MIPS, um PowerPC e um Sparc, uma plataforma com 8 *cores* possui dois ARM, dois MIPS, dois PowerPC e dois Sparc, e assim sucessivamente. Além disso, as plataformas heterogêneas rodam apenas um software muito específico, desenvolvido especialmente para elas. Devido a essas características peculiares, trechos de código específicos tiveram de ser incluídos no script do *MPSoCBench* de modo que as plataformas heterogêneas pudessem ser geradas e executadas pelo mesmo. Foram criadas funções de `build`, `run` e geração de Makefile específicas para tais plataformas. Além disso, foi introduzido um argumento específico para as mesmas no script.

Em paralelo a esses problemas descritos, uma questão sempre presente foi a passagem de argumentos para o script. Os argumentos foram várias vezes modificados ao longo da implementação do *benchmark*, devido tanto a algumas alterações no próprio escopo do projeto quanto a maneiras mais inteligentes e

intuitivas de passagem de argumentos. Desde o começo o *parsing* dos argumentos foi feito de forma própria pelo script, sem a utilização de bibliotecas para facilitar o *parsing* dos mesmos. Contudo, à medida que os argumentos foram ficando complexos, o orientador sugeriu que fosse utilizada a biblioteca do *Python* `optparse` para facilitar a passagem e interpretação dos argumentos. Contudo, ao estudar o uso da biblioteca, o aluno notou que a mesma era considerada *deprecated* pelo *Python*, sugerindo o uso da biblioteca `argparse`. Apesar disso, a biblioteca `argparse` não era exatamente adequada para a proposta de passagem de argumentos do *MPSoCBench*. Mais precisamente, seria necessário redefinir muitos métodos da biblioteca através de derivação da mesma em uma biblioteca de *parsing* de argumentos personalizada para o script. Em última análise, o *overhead* de fazer o *parsing* no próprio código ("na unha"), sem extender a biblioteca argparse era consideravelmente menor do que o *overhead* de herdar a `argparse` e redefinir alguns métodos. Sendo assim, o *parsing* dos argumentos é feito de forma particular ao script de *frontend* do *MPSoCBench*, sem usar bibliotecas de terceiros. Vale ressaltar que o *parsing* de argumentos no *ACPM* é feito de forma semelhante, sem utilizar `argparse`, `optparse` ou qualquer outra biblioteca.

Depois desses esforços no *MPSoCBench*, algumas coisas puderam ser aproveitadas no *ACPM*. A principal delas foi a estruturação dos Makefiles, que ficou organizada da mesma forma que foi feita no *benchmark*. Dividiu-se o Makefile original do *ARP* em três, assim como no *benchmark*.

4 Resultados

Até agora tivemos com principal resultado a primeira versão do *ACPM*, que está disponível no site do *ArchC* (archc.org) para download, assim como a versão final do script de *frontend* do *MPSoCBench*, e a versão final do *MPSoCBench* como um todo.

Um paper sobre o *MPSoCBench* foi submetido ao periódico do IEEE "Embedded Systems Letters", tendo sido retornado para revisões, mas ultimamente rejeitado.

Após a rejeição do artigo pela "Embedded Systems Letters", o mesmo foi submetido ao NASCUG (North American SystemC User's Group) para apresentação, tendo sido aceito. O NASCUG é um encontro de usuários de *SystemC* que ocorrereu durante a DAC (Design Automation Conference), no mês de junho no Texas. A Liana fez a apresentação do *MPSoCBench*.

O paper sobre o *MPSoCBench* foi submetido ao SBAC-PAD 2013 e atualmente está sendo analisado.

5 Discussão / Conclusões

É extremamente notável o aprendizado acumulado até agora nesta atividade de Iniciação Científica. Sem dúvida, a parcela mais significativa do aprendizado foi relacionada à boas práticas de programação e em segundo lugar à documentação da atividade como um todo. Além disso, houve um aprofundamento significativo no conhecimento da linguagem *Python*, que mesmo o aluno já possuindo um conhecimento razoável da mesma, conseguiu expandir ainda mais

seus conhecimentos, tornando-se praticamente fluente nessa útil ferramenta.

O grande aprendizado de boas práticas de programação se deu por conta do projeto exigir que código de alta qualidade fosse escrito. Além disso, o trabalho de estudo do código já existente contribuiu bastante para estabelecer o nível de qualidade exigido pelo projeto. Não que o código original fosse ruim, mas alguns detalhes passaram despercebidos. Contudo, estas pequenas falhas foram detectadas e corrigidas, resultando em uma ligeira melhora da qualidade do código. Além disso, as melhorias propostas e implementadas aumentaram ainda mais o nível do código como um todo. Além disso, alguns problemas interessantes foram solucionados, contribuindo para o aprendizado e o aumento das habilidades de programação do aluno e resultaram num maior conhecimento da linguagem *Python* por parte do aluno.

Quanto à documentação, esta foi imposta por iniciativa própria do aluno, como forma de organização pessoal. Um reflexo disso é que sempre era tomada nota de todas as reuniões realizadas, tanto com o orientador quanto com a aluna de doutorado. Desse modo, sempre se tinha uma noção das tarefas a serem realizadas e também de continuidade da atividade de Iniciação Científica como um todo. Outro aspecto positivo que colaborou para a devida documentação da atividade foi a escolha da ferramenta *Git* como controle de versão.

Finalmente, pode-se afirmar que o saldo da Iniciação foi extremamente positivo. Além disso, é bastante significativo o aprendizado do aluno ao longo de toda a atividade até agora.

Referências

[1] Rodolfo Azevedo, Bruno Albertini, and Sandro Rigo. Arp: Um gerenciador de pacotes para sistemas embarcados com processadores modelados em archc. In *Workshop de Sistemas Embarcados - WSE*. SBC, 2010. In Portuguese.

[2] Rodolfo Azevedo, Sandro Rigo, Marcus Bartholomeu, Guido Araujo, Cristiano Araujo, and Edna Barros. The archc architecture description language and tools. *International Journal of Parallel Programming*, 33:453–484, 2005. 10.1007/s10766-005-7301-0.

[3] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The parsec benchmark suite: characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 72–81. ACM, 2008.

[4] Jason Clemons, Haishan Zhu, Silvio Savarese, and Todd Austin. Mevbench: A mobile computer vision benchmarking suite. In *Workload Characterization (IISWC), 2011 IEEE International Symposium on*, pages 91–102. IEEE, 2011.

[5] Matthew R Guthaus, Jeffrey S Ringenberg, Dan Ernst, Todd M Austin, Trevor Mudge, and Richard B Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pages 3–14. IEEE, 2001.

[6] Syed Muhammad Zeeshan Iqbal, Yuchen Liang, and Håkan Grahn. Parmibench-an open-source benchmark for embedded multiprocessor systems. *Computer Architecture Letters*, 9(2):45–48, 2010.

[7] Chunho Lee, Miodrag Potkonjak, and William H Mangione-Smith. Mediabench: a tool for evaluating and synthesizing multimedia and communicatons systems. In *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*, pages 330–335. IEEE Computer Society, 1997.

[8] Stan Liao, Grant Martin, Stuart Swan, and Thorsten Grötker. *System design with SystemC*. Kluwer Academic Pub, 2002.

[9] S. Rigo, G. Araujo, M. Bartholomeu, and R. Azevedo. Archc: a systemc-based architecture description language. In *16th Symposium on Computer Architecture and High Performance Computing, 2004 - SBAC-PAD 2004*, pages 66 – 73, 2004. Best Paper Award.

[10] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The splash-2 programs: Characterization and methodological considerations. In *ACM SIGARCH Computer Architecture News*, volume 23, pages 24–36. ACM, 1995.