PPP Debugger - Ambiente para Depuração de Programas do Paradigma Paralelo

VINÍCIUS GARCIA PERUZZI; RAFAEL BURLAMAQUI AMARAL
Universidade Federal de Pelotas – vgperuzzi@inf.ufpel.edu.br
Universidade Federal de Pelotas – rafael.amaral@inf.ufpel.edu.br

1. INTRODUÇÃO

O paradigma da programação paralela vem cada vez crescendo e se tornando mais popular na comunidade de desenvolvimento de softwares em geral, isso se dá pela busca do aumento da capacidade computacional para resolver problemas complexos e mesmo para manipulação de grandes volumes de dados que surgem por necessidades modernas, por exemplo, o tratamento de preferências para um sistema de recomendação de um serviço de *streaming*.

Com isso, novas práticas, ferramentas e paradigmas precisam ser desenvolvidas para que se possa alcançar um maior desempenho no desenvolvimento de um software, segundo (SHALF, 2020), aproveitar de programas cada vez mais limpos e robustos é uma abordagem fundamental para alcançar o futuro esperado da computação, especialmente em um início de época pós Lei de Moore (MOORE, 2006).

No desenvolvimento de programas, o processo de depuração é, sempre, uma prática recomendada, e, caso aplicada, concede níveis de robustez de código e desempenho que não seriam possíveis caso não houvesse. Nesse contexto, O GDB (*GNU Debugger*) é uma ferramenta difundida e conhecida pelos programadores das linguagens C, C++, GO, D, Rust, Pascal, Assembly e outras (STALLMAN; PESCH; SHEBS, 2017).

Entretanto, pensando nas necessidades específicas e características da programação paralela, é possível aprimorar a usabilidade do GDB, que ainda se dá por linha de comando no terminal, fazendo uso de um ambiente de depuração, apoiado sobre ele (O GDB), que tenha foco na depuração dos programas desse paradigma, concedendo ao desenvolvedor a abstração necessária para atingir programas mais robustos de forma mais simplificada.

Portanto, apresenta-se o *PPP Debugger (Plush Puppy Parallel Debugger)*, desenvolvido com o objetivo de ser o ambiente de depuração citado, aprimorando não só a usabilidade do GDB, mas também concedendo uma visualização mais direcionada à visualização e troca de contextos de diferentes *threads* lançadas por programas deste paradigma. O desenvolvimento desta ferramenta é, também, a pesquisa e trabalho de conclusão de curso do aluno Vinícius Garcia Peruzzi, no curso de ciência da computação na UFPel.

2. METODOLOGIA

O trabalho se deu em três principais fases: primeiro a fase de estudos de pertinência e viabilidade, e levantamento do estado da arte e trabalhos relacionados, depois a fase de projeto da ferramenta e estudo das tecnologias necessárias para o desenvolvimento, e por fim o desenvolvimento da ferramenta.

Na primeira etapa, uma interface gráfica simples foi implementada, mas que já trazia a integração de um *layout* gráfico com as funcionalidades do GDB, via biblioteca encontrada na pesquisa.

A biblioteca em questão: *pygdbmi,* foi desenvolvida para uso no GDBGui (SMITH, 2024), implementada pelo próprio desenvolvedor da ferramenta, na qual uma conexão com a interface de máquina do GDB é instanciada, e, a partir de então, é possível submeter comandos e receber respostas. O que era feito através de uma interface gráfica implementada com a PySide6 do Python, que permite o desenvolvimento de interfaces gráficas mais robustas e dinâmicas, concedendo portabilidade aos sistemas mais populares do mercado.

Nessa etapa também foi realizado um estudo sobre as funcionalidades e características do DDD, que é um depurador de código com interface gráfica de propósito geral, que visa a visualização do processo de depuração. Entretanto, ele não possui uma boa abstração quanto à usabilidade, não entregando seus comandos de forma muito mais abstraída do que a própria interface de linha de comando do GDB, na verdade, a proposta dele é poder visualizar os pontos de paradas e estruturas de dados de forma mais avançada. Pois suas funcionalidades geralmente estão dispostas em submenus que fazem necessários quatro cliques ou mais para serem alcançadas, deixando muito a desejar na parte referente à praticidade e agilidade do processo de depuração.

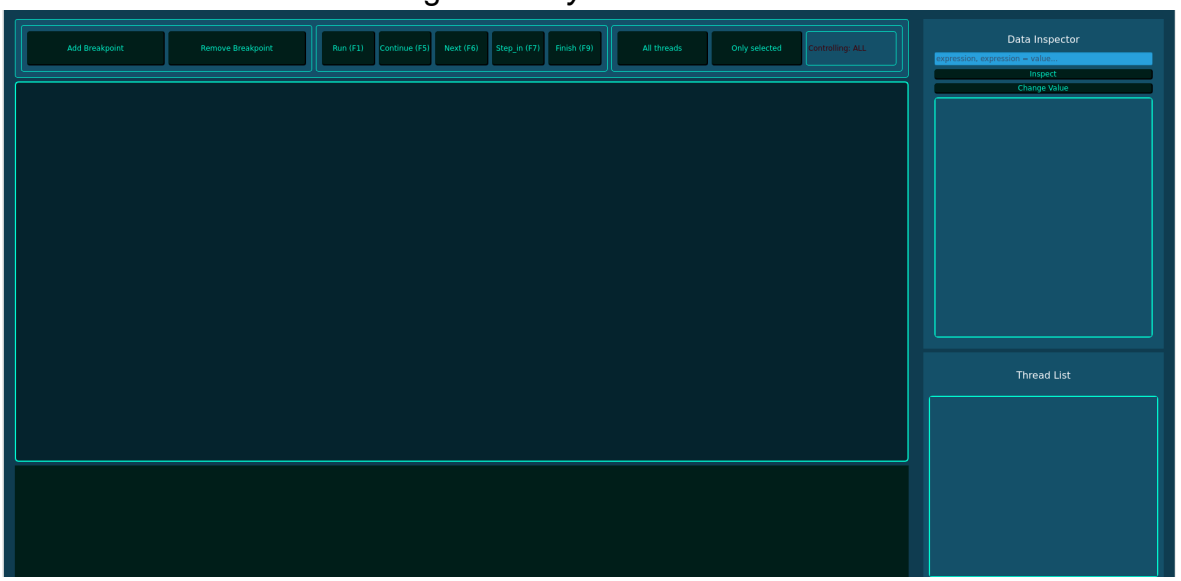
Na segunda etapa, foi quando todo o processo de planejamento da ferramenta ocorreu, levantando requisitos funcionais e não funcionais, os quais foram dispostos no formato de *user stories*, gerando desta forma um diagrama de classes e um diagrama da arquitetura geral da ferramenta.

E por fim, o desenvolvimento trouxe à tona o ambiente de depuração, inicialmente com uma versão *alpha*, que já permite experimentar a usabilidade e comparar com o próprio GDB e outras ferramentas gráficas para depuração.

3. RESULTADOS E DISCUSSÃO

O *layout* do PPP foi projetado para suprir as demandas da programação paralela. A Figura 1 expõe o *layout* básico.

Figura 1 - *layout* básico do PPP

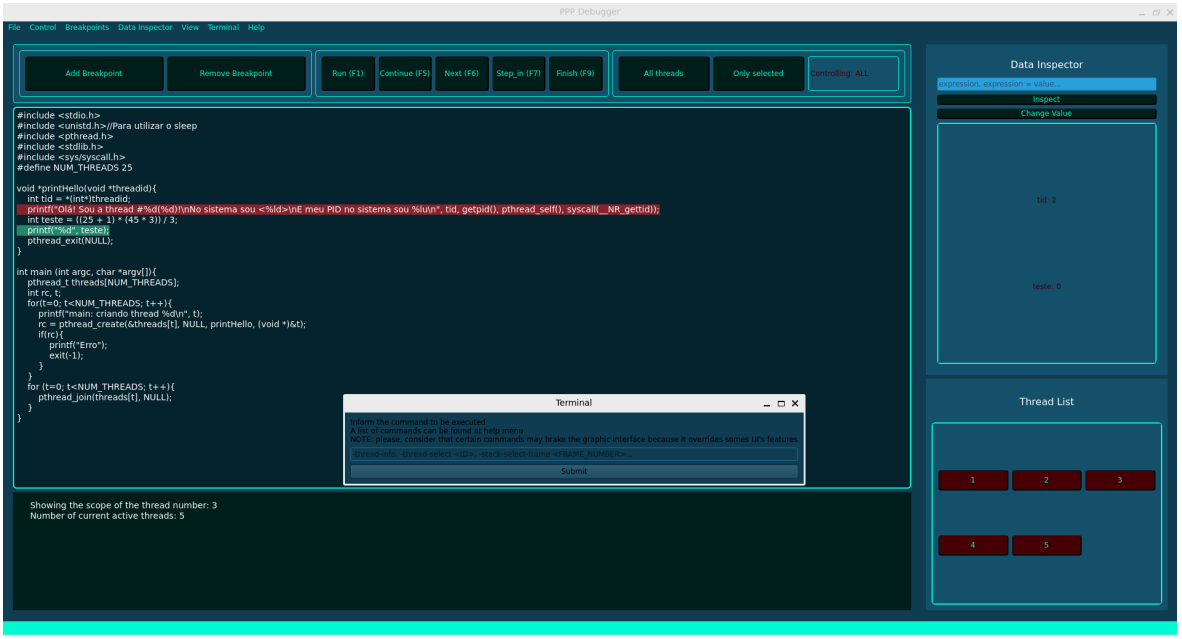


O PPP possui três docas superiores, duas com os comandos mais utilizados em qualquer processo de depuração, e outra com a funcionalidade de poder travar a execução de outras *threads* e focar a execução apenas na selecionada no momento, o que antes era um comando tido como avançado na depuração com o GDB, na interface é tão simples quanto o clique de um botão.

Também dispõem de uma região onde expressões podem ser inseridas para que fiquem sendo monitoradas em tempo real, sendo sensíveis aos avanços de execução ou troca de contexto de *threads*, ou seja, isso significa que ao adicionar uma expressão e alterar o foco entre distintos fluxos de execução, esses valores serão igualmente atualizados, sem a necessidade do usuário reinserir os valores, ou enviar o comando de impressão de cada um dos valores para cada uma das *threads* como seria no processo tradicional usando GDB.

Ainda, existe uma região onde a quantidade de *threads* ativas é constantemente exibida para o usuário, não apenas mostrando a quantidade e o identificador de cada uma delas, mas também sendo cada uma um botão que, ao ser clicado, realiza a troca de escopo para o contexto da *thread* selecionada.

Figura 2 - código carregado com breakpoints, *threads* e terminal



A Figura 2 mostra o código do usuário carregado na ferramenta, com alguns pontos de paradas inseridos, para inseri-los, basta simplesmente pousar o cursor sobre a linha desejada e clicar no botão referente, ou então na função no menu superior. Para remover utiliza-se o mesmo procedimento.

Como a ferramenta tem a proposta de ser um ambiente de depuração completo, o código pode ser alterado dentro da própria ferramenta, salvo, compilado e usado para iniciar um novo processo de depuração, sem a necessidade de utilizar quaisquer ferramentas externas, ou sair da ferramenta para reiniciar uma nova depuração.

Como dito anteriormente, o foco do *layout* visa suprir as necessidades da depuração paralela de forma simples e prática, porém, caso o usuário queira utilizar comandos que não estejam mapeados em botões na interface, ou em funcionalidades no menos superiores, o que pode-se considerar uma utilização avançada da ferramenta, é possível iniciar um terminal de comandos, onde comandos de interface de máquina e do próprio GDB podem ser inseridos manualmente como seriam na interface de linha de comando, que serão tratados e executados de forma adequada, e ainda é possível o programador tirar proveito da interface gráfica para visualização do estado do programa.

Por fim, considerada uma prática ainda mais avançada no processo de depuração, a ferramenta ainda possibilita a visualização do código de máquina

(assembly) sendo executada na função em execução da *thread* selecionada atualmente, bem como a lista de registradores da *thread* em questão.

Sobre a usabilidade do PPP, pode-se discutir que ele aprimora bastante o processo de depuração do GDB, especialmente considerando o contexto da programação paralela, onde existem muitas ações repetitivas. Por exemplo: no GDB, caso se queira analisar o valor de 3 expressões no contexto de todas as *threads*, supondo que sejam 3 em determinado momento, é necessário usar um comando para listar as *threads* ativas no momento, usar outro comando para selecionar uma delas e, então, para cada expressão que se deseja verificar usar um outro comando que faz a impressão dos valores, então mudar o escopo com outro comando para outra *thread* e repetir todo o processo. Já no PPP, basta adicionar a expressão ao inspecionador que para qualquer alteração que aconteça, o valor já é atualizado automaticamente, o que se mostra especialmente valioso ao alternar entre *threads*.

Outro exemplo é a simplicidade para editar o código e salvar, recomeçando o processo, o qual exigiria o fechamento do GDB, a utilização de um editor externo, a recompilação do programa e o reinício do GDB, o que acontece da mesma forma no PPP, mas todo o processo é realizado automaticamente ao salvar.

Já com relação ao DDD, o aprimoramento da usabilidade se dá em torno do foco no processo da depuração paralela, já que as funcionalidades para isso são pouco intuitivas do DDD, e mesmo a edição de código só é possível dentro da ferramenta, pois ela permite o acesso via plugin de outros editores de texto como o VIM, ou o Nano, ainda exigindo que o usuário os tenha instalados.

A troca de *threads* no DDD se dá por uma funcionalidade no submenu *source*, o que faz com que a troca de *threads* não seja tão prática quanto o que se faz necessário no processo de depuração paralela, nos quais, facilmente, dez ou doze *threads* serão lançadas, ainda mais considerando os hardwares modernos.

4. CONCLUSÕES

Baseado no processo metodológico descrito, nos resultados apresentados e nas comparações discutidas, pode-se dizer que a ferramenta supre bem a necessidade a qual se propõe, entregando ao programador do paradigma paralelo uma opção de aprimorar a usabilidade do já conhecido e consolidado GDB, permitindo mais velocidade no processo de depuração, com informações mais aparentes e dinâmicas e funcionalidades mais acessíveis e práticas.

5. REFERÊNCIAS BIBLIOGRÁFICAS

MOORE, G. E. Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff. IEEE Solid-State Circuits Society Newsletter, [S.l.], v.11, n.3, p.33–35, 2006.

SHALF, J. The future of computing beyond Moore's Law. Philosophical Transactions of the Royal Society A, [S.l.], v.378, n.2166, p.20190061, 2020.

SMITH, C. A browser-based frontend to gdb (gnu debugger). Acesso em 09 mar. 2024. Disponível em: <https://www.gdbgui.com/>.

STALLMAN, R.; PESCH, R.; SHEBS, S. Debugging with gdb. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA: Free Software Foundation, 2017.

ZELLER, A.; LüTKEHAUS, D. DDD A Free Graphical Front-End for UNIX Debuggers. ACM SIGPLAN Notices, [S.l.], v.31, n.1, p.22–27, 1996.