Paralelização do compilador da linguagem de programação Rust

Yuri Kunde Schlesner

5 de agosto de 2014

1 Identificação

Resumo: TODO.

Período de execução: Setembro de 2014 a Dezembro de 2014

Unidades participantes:
Curso de Ciência da Computação
Departamento de Eletrônica e Computação

Área de conhecimento: Ciência da Computação

Linha de Pesquisa: Compiladores, Programação Paralela

Tipo de projeto: Trabalho de Conclusão de Curso

Participantes:
Profª Andrea Schwertner Charão – Orientadora
Yuri Kunde Schlesner – Orientando

2 Introdução

Um dos principais riscos presentes quando se opera um sistema de computadores é que ele seja atacado por invasores maliciosos. Embora alguns ataques sejam permitidos pela má configuração de servidores e sistemas de segurança, uma significante parte é realizada explorando falhas de programação no software rodando nesses servidores. Erros como *buffer* ou *stack overflows*, acesso a ponteiros nulos e sincronização inadequada entre *threads*, assim como outros tipo de erros de

memória, permitem um atacante manipular e acessar informações que não deveria, alterando o funcionamento do software.

Essas e muitas outras classes de erros só são possíveis devido a escolha de C e C++ de permitir acessos não-checados a memória. Outras linguagens, como Java, Python e inúmeras outras linguagens de alto-nível, fizeram a escolha de sacrificar performance e controle para eliminar estes tipos de erro, forçando o uso de *garbage collection* e acessos checados a arrays.

No entanto, existem contextos em que o uso dessas técnicas é considerado inaceitável, como por exemplo no desenvolvimento de *kernels* ou *drivers* para sistemas operacionais; em aplicações de tempo real, como automação e controle ou sistemas de prevenção de acidentes; ou aplicações que requerem alta performance, como computação científica, servidores que precisam responder a milhares de requisições por segundo ou jogos. Estes tipos de tarefas geralmente são intituladas "programação de sistemas".

A linguagem de programação *Rust*, um projeto de pesquisa da *Mozilla Research*, tem como objetivo preencher este nicho como uma linguagem de programação de sistemas que foca simultaneamente em alta-performance, segurança e expressividade. Ela atinge isso usando um modelo tradicional de compilação prévia (*ahead of time*) e um sistema de tipos que permite a verificação automática dos usos de ponteiros durante a compilação, eliminando a possibilidade de acontecerem erros de memória, mas sem introduzir penalidades excessivas de performance ou consumo de memória.

3 Objetivos

3.1 Objetivo Geral

O objetivo geral deste trabalho é realizar otimizações e re-estruturar o *rustc*, o compilador oficial de *Rust*, de forma que ele faça melhor uso de processadores com múltiplos núcleos e reutilize o trabalho de compilações anteriores, afim de reduzir o tempo necessário para a compilação de programas.

3.2 Objetivos Específicos

- Estudar a arquitetura interna do compilador e identificar quais módulos podem realizar seu trabalho independentemente.
- Realizar modificações que separem o processo de compilação em partes, e executem estas partes simultaneamente.

- Se mostrar-se viável, fazer o compilador detectar etapas para as quais os dados de entrada não mudaram, e portanto não precisam ser re-processadas.
- Realizar medições de performance para avaliar os ganhos de tempo atingidos pelas modificações.
- Caso estas mostrem-se vantajosas, integrar as mudanças na versão oficial do compilador.

4 Justificativa

Normalmente, um compilador utilizar apenas um núcleo não é um grande problema, pois cada arquivo fonte é compilado separadamente e portanto basta invocar várias instâncias do compilador simultâneamente para processar cada arquivo. Os objetos intermediários gerados são então passados por um *linker*, que os úne para produzir um binário final, num processo geralmente mais simples, e portanto mais rápido, que a compilação do mesmos.[1]

A linguagem *Rust* diverge deste modelo de compilação e compila um programa ou biblioteca inteira como uma única unidade. Isto é necessário pois não existe uma forma externa das declarações do programa (como arquivos *header* em C), então todo o código fonte precisa ser lido na mesma invocação do compilador para descobrir os tipos e funções presentes. No momento, todo o processo é feito de forma serial.

Para ajudar com a adoção de uma linguagem, é importante que ela seja atrativa para potenciais programadores que estejam buscando uma oportunidade de se livrar de dificuldades que já encontraram em outras linguagens, e rápidos ciclos de iteração, atingidos através duma diminuição do tempo de compilação, são um fator importante a se considerar para atingir esse objetivo. (Longos tempos de compilação são uma crítica frequentemente dirigida a C++, por exemplo.) Portanto, este trabalho ajudará a adoção de *Rust* em projetos maiores que enfrentariam problemas com tempos de compilação.

5 Revisão de Literatura

TODO.

[1] O processo de *linking* é, no entanto, também uma etapa demorada em projetos de larga escala com muitos objetos intermediários. Por esse motivo, alguns *linkers* possuem parallelização interna deste processo.

6 Metodologia

Dado seu caráter prático, de resolver um problema específico em um determinado domínio, esta pesquisa se enquadra como pesquisa aplicada. Como o objetivo será de explorar o possível espaço de soluções arquiteturais para o problema, e depois documentar quais destas mostraram-se adequadas, é uma pesquisa exploratória.

7 Plano de Atividades e Cronograma

1. **Estudar a arquitetura interna do compilador:** Primeiro será feito um levantamento da estrutura geral do compilador, afim de identificar o trabalho necessário para e quais as tarefas que podem ser paralelizadas. Caso fique evidente que serão necessárias grandes mudanças na arquitetura do compilador, estas também serão planejadas com antecedência.

2. **Divisão/paralelização da compilação:** Aqui será realizado o trabalho de implementação, além da finalização dos detalhes arquiteturais de mais baixo nível, para permitir que o compilador utilize vários núcleos do processador simultâneamente. Este trabalho fará uso do conhecimento adquirido e do planejamento realizado na etapa 1.

3. **Re-utilização de compilações anteriores:** Será adicionado ao compilador a capacidade de re-utilizar resultados de compilações anteriores nos quais os arquivos fonte que influenciem a compilação não tenham sofrido alterações que afetem o resultado. Aqui será novamente utilizado as informações coletadas na etapa 1.

4. **Medições de performance:** Para averiguar o impacto das mudanças efetuadas nas etapas 2 e 3, serão feitas medições de tempo e de performance (utilizando um *profiler*). Esta etapa contribuirá uma medição quantitativa do sucesso obtido nesta pesquisa em alcançar seus objetivos.

5. **Integração das mudanças:** Afim de beneficiar a comunidade geral de *Rust*, as modificações efetuadas serão submetidas devolta ao projeto, para aprovação dos membros do time de desenvolvimente e subsequente integração. Espera-se que esse seja um processo contínuo, com pedidos de comentários dos desenvolvedores afim de maximizar a qualidade do trabalho.

Espera-se que o desenvolvimento das atividades siga o seguinte cronograma:

Etapa	Setembro	Outubro	Novembro	Dezembro
1	✓			
2	✓	✓		
3			✓	✓
4	✓	✓	✓	✓
5		✓		✓

Tabela 1: Cronograma de Atividades

8 Recursos

Para a realização deste trabalho será utilizado apenas equipamento pessoal do pesquisador, visto que não é necessário o uso de qualquer equipamento especial além de um computador para desenvolvimento.

9 Resultados Esperados

Ao término deste trabalho, espera-se que seja disponibilizada uma versão do compilador de *Rust* para realizar a compilação de programas de forma paralela, utilizando vários núcleos de CPU, e re-utilizando trabalho de compilações anteriores, de forma que esta seja completada em menos tempo. Caso os resultados, baseados nas medições de performance realizadas, mostrem-se suficientemente vantajosos, espera-se que as mudanças presentes nessa versão sejam integradas na versão oficial do compilador, de forma a beneficiar todos os membros da comunidade de usuários de *Rust*.

10 Referências

TODO.