DeLIAP e DeLIAJ: Interfaces de biblioteca de dependabilidade em PAD para Python e Julia

Marcos Irigoyen[1], **Carla Santana**[2], **Ramon C. F. Araújo**[3], **Samuel Xavier-de-Souza**[1]

[1] Departamento de Engenharia de Computação e Automação
Universidade Federal do Rio Grande do Norte (UFRN) – Natal, RN – Brazil

[2]Laboratório de Arquiteturas Paralelas para Processamento de Sinais
Universidade Federal do Rio Grande do Norte (UFRN)

[3]Departamento de Física Teórica e Experimental
Universidade Federal do Rio Grande do Norte (UFRN)

{marcos.irigoyen.706,carla.santana.058}@ufrn.edu.br,

ramon@fisica.ufrn.br, samuel@dca.ufrn.br

Resumo. *Para suprir a demanda de recursos computacionais em sistemas de processamento de alto desempenho contemporâneos, é comum o escalonamento horizontal de componentes. Em contrapartida, cada novo componente é um potencial ponto de falha, fazendo essencial o emprego de técnicas de tolerância a falhas. Nesse contexto, a biblioteca de tolerância a falhas DeLIA foi desenvolvida em C++ com capacidades de detecção e recuperação de falhas. Neste trabalho propomos estender as capacidades da DeLIA para Python e Julia através das interfaces DeLIAP e DeLIAJ e divulgamos resultados preliminares de seu uso para um programa de imageamento sísmico em Julia com cálculo do custo adicional mediano (1,4%) e relato de implementação.*

1. Introdução

A crescente complexidade de novos problemas promove a necessidade de cada vez mais recursos computacionais. Hoje em dia, sistemas computacionais para Processamento de Alto Desempenho (PAD) são sistemas distribuídos, evoluindo em desempenho e quantia de máquinas para atender a demanda de programas paralelos [Silva et al. 2022]. Esta natureza distribuída chama, per se, por precauções no desenvolvimento e execução de tais programas, seja para assegurar-se da corretude, operacionalidade e disponibilidade da aplicação, ou minimizar possíveis perdas. Cada componente discreto de um sistema é suscetível a falhas, que, independentemente da origem, podem causar disrupções na sua ocorrência. Junto a maior severidade de falhas em PAD, dado o inerentemente maior volume de trabalho possível de se perder; e o aumento do risco de falhas pelo uso massivo de nós computacionais, onde cada nó é um potencial ponto único de falha em aplicações sem as devidas precauções, o emprego de técnicas de tolerância a falhas é imperativo.

Para facilitar a integração de tais técnicas, a biblioteca de confiabilidade para aplicações iterativas DeLIA (do inglês *Dependability Library for Iterative Applications*) foi proposta com capacidades de detecção de interrupção e recuperação de falhas [Santana et al. 2024]. Aqui, propõe-se estender as suas funcionalidades para Python (DeLIAP) e Julia (DeLIAJ), linguagens bem estabelecidas em computação científica em geral

que, para além da maior facilidade de uso e prototipagem, não dispõem da mesma gama de soluções de tolerância a falhas que as linguagens mais convencionais a PAD possuem.

Pondo em perspectiva, as soluções desvinculadas a modelos de paralelização que atuem a nível de aplicação em Python estão na forma de versões especializadas para os *frameworks* da aplicação ao qual são desenvolvidas, como detecção de falhas em análise de genomas na xGAP[Gorla et al. 2021] e salvamento de parâmetros em bibliotecas de aprendizado profundo; enquanto que, para Julia, o mais próximo de alternativas encontradas foram as capacidades de integração de soluções próprias em modelos de paralelização.

Diante disso, as interfaces facilitam a inclusão de tolerância a falhas em aplicações gerais de PAD, sem dependência do protocolo de paralelismo ou *framework* da aplicação; enquanto dispõem de capacidades de salvamento, detecção de sinais de terminação e monitoramento por batimento que não são encontradas em uma única biblioteca mesmo em C/C++ [Santana et al. 2024].

Para validar a eficácia da solução proposta, aplicamos a interface em Julia em uma base de códigos no método de inversão de onda completa da forma de onda 4D (FWI 4D, do inglês *Full Waveform Inversion* 4D), utilizada internamente no grupo de pesquisa do projeto "Novas metodologias computacionalmente escaláveis para sísmica 4D orientado ao alvo em reservatórios do pré-sal"[Carvalho et al. 2021]. Foram feitas análises do custo computacional introduzido pela inclusão da biblioteca e relato da implementação.

Após menção mínima da teoria, separamos a metodologia para abordar como foi feito o experimento de custo computacional e a implementação das interfaces nas linguagens-alvo. Os resultados e discussão ocupam a seção seguinte da mesma forma.

1.1. Embasamento Teórico

Apesar da ampla teoria e modelagem de diversos tipos de falha e seus comportamentos, convém à brevidade abordar somente o modelo principal ao qual a DeLIA lida: o modelo *fail-stop*, que representa a interrupção completa da execução do componente faltoso da aplicação na ocorrência da falha, evitando interação com o resto do sistema, que detecta a falha na componente independentemente.

Como um todo, tolerância a falha designa-se à capacidade de um sistema continuar operando adequadamente apesar da ocorrência de falhas, podendo ser dividida (em concepções mais gerais) em detecção e recuperação de erros, confinamento, e tratamento da falha[Weber 2003]. A DeLIA abrange a detecção e recuperação de erros, enquanto que o confinamento é pressuposto no modelo *fail-stop* e o tratamento da falha é feito à parte, dependendo da natureza da falha e do conhecimento de sua causa.

Para detecção de erros a biblioteca provê detecção de sinais de terminação e monitoramento de batimentos, enquanto que para recuperação de erros se usam pontos de salvamento. Para garantir a consistência dos salvamentos a DeLIA se aproveita da natureza de aplicações no modelo *Bulk Synchronous Parallel* [Valiant 1990], mais especificamente na constituição de sua computação por etapas globais e locais, onde entre cada etapa global há um conjunto de tarefas para ser executado localmente, com possibilidade de comunicação entre componentes. Na conclusão do conjunto de tarefas, sincroniza-se o estado global da aplicação e segue-se à próxima etapa global. Ao fim de uma etapa global é possível de se encaixar um salvamento global, enquanto que, assumindo independência

entre nós nas tarefas da etapa local, permite salvamento do estado local.

O FWI 4D, aplicação do estudo de caso, é uma técnica de imageamento sísmico usada para monitoramento de reservatórios, onde através da aplicação do FWI a dados coletados entre um intervalo de tempo é possível estimar as mudanças no reservatório neste intervalo. O FWI por si é um algoritmo iterativo de inversão de onda completa baseado na estimativa de um modelo físico que produza um sismograma próximo do sismograma obtido experimentalmente para uma entrada conhecida. Devido a massiva escala de dados usados na prática, a aplicação tem um alto custo computacional nos dias de hoje, com falhas durante a sua execução causando contratempos de semanas a meses.

2. Metodologia

O desenvolvimento das interfaces se deu pelo uso de Cython e CxxWrap para a interoperabilidade da biblioteca com Python e Julia; para verificar o comportamento foram feitos programas paralelos de testes para cada funcionalidade, usando mpi4py e o módulo Distributed; foram usados Doxygen para documentação e git para versionamento, com ambas as interfaces disponíveis no repositório da DeLIA, de código aberto, no GitLab.

O estudo de caso é da aplicação da DeLIAJ para uma base de código ativamente usada por pesquisadores no supercomputador do Núcleo de Processamento de Alto Desempenho (NPAD) da UFRN. Para desenvolvimento e testagem mais ágeis, foi desenvolvido um código exemplo para FWI 4D em uma carga menor à habitual. Como a complexidade computacional de um salvamento é linear com o tamanho dos dados, podemos assumir que resultados para tamanhos de problema maiores terão custo computacional relativo igual ou menor aos obtidos aqui, dada uma mesma configuração. Por motivos expressos no relato de implementação junto aos resultados, foram usadas somente as capacidades de salvamento global e detecção de sinais de terminação, com o primeiro realizando salvamentos a cada iteração global.

Aquisição e tratamento dos dados levaram 3 dias, com 10 execuções de 20 iterações do algoritmo para um conjunto de 50 amostras com e sem a DeLIA, distribuídos em 32 núcleos de um nó no NPAD. Dessas execuções foram extraídos os tempos de execução para cálculo do custo computacional mediano relativo e elaboração de um gráfico de box-plot das distribuições, similarmente a [Santana et al. 2024].

3. Resultados e Discussão

A aplicação do estudo de caso teve funcionamento adequado, mas especificidades na forma que o paralelismo era empregado na aplicação e na atomicidade das funções da base de código restringiram as funcionalidades possíveis de se aproveitar na aplicação.

Em questão de desempenho, o custo computacional relativo estimado ficou em torno de 1,4%, com valor absoluto de aproximadamente 174,9448 segundos para os 13441,8312 segundos medianos da execução com a biblioteca. Pelo gráfico na figura 1 é possível ver os dois *box-plots* das amostras em uma mesma escala para comparação, onde, pela ordem da escala podem se considerar próximos.

4. Conclusão

A fim de tornar tolerância a falhas mais acessível em PAD foram propostas e desenvolvidas interfaces da biblioteca DeLIA para Julia (DeLIAJ) e Python (DeLIAP), estendendo

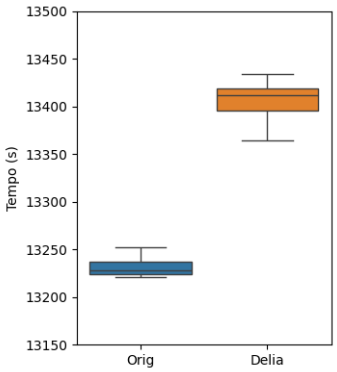

Figura 1. *Box-plots* das amostras coletadas.

as capacidades da biblioteca base para linguagens mais fáceis e ainda bem estabelecidas em computação científica como um todo. Cada funcionalidade mencionada no artigo da DeLIA [Santana et al. 2024], exceto onde limitado pelas linguagens-alvo, estão disponíveis, sendo verificadas por testes automatizados.

Buscou-se validar a eficácia das interfaces para prover tolerância a falhas em aplicações PAD através da integração da DeLIAJ em uma base de códigos de FWI 4D em Julia. Nos testes, obteve-se um custo computacional mediano relativo de aproximadamente 1,4%, favorável à eficiência computacional da interface. Como contraponto, fatores da aplicação impossibilitaram o uso de parte das funcionalidades da biblioteca. Por fim, consideramos as interfaces viáveis para utilização externa, apesar das reconhecíveis limitações a serem resolvidas.

Referências

Carvalho, P. T. C., da Silva, S. L. E. F., Duarte, E. F., Brossier, R., Corso, G., and de Araújo, J. M. (2021). Full waveform inversion based on the non-parametric estimate of the probability distribution of the residuals. *Geophysical Journal International*, 229(1):35–55.

Gorla, A., Jew, B., Zhang, L., and Sul, J. H. (2021). xgap: a python based efficient, modular, extensible and fault tolerant genomic analysis pipeline for variant discovery. *Bioinformatics*, 37(1):9–16.

Santana, C., Araújo, R. C., Sardina, I. M., Ítalo A.S. Assis, Barros, T., Bianchini, C. P., de S. Oliveira, A. D., de Araújo, J. M., Chauris, H., Tadonki, C., and de Souza, S. X. (2024). Delia: A dependability library for iterative applications applied to parallel geophysical problems. *Computers & Geosciences*, 191:105662.

Silva, G., Bianchini, C., and Costa, E. (2022). *Programação Paralela e Distribuída com MPI, OpenMP e OpenACC para computação de alto desempenho*. Aovs Sistemas de Informática.

Valiant, L. G. (1990). A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111.

Weber, T. S. (2003). Tolerância a falhas: conceitos e exemplos. *Apostila do Programa de Pós-Graduação–Instituto de Informática-UFRGS. Porto Alegre*, page 24.