Incentivando a adoção de modelos de programação concorrente/paralela a partir da visualização de ganhos na geração de fractais em tempo-real*

Lucas Morais†, Rafael Torchelsen, Gerson Geraldo H. Cavalheiro

Programa de Pós-graduação em Computação
Centro de Desenvolvimento Tecnológico
Universidade Federal de Pelotas
Pelotas – RS – Brasil

{lzmorais,rafael.torchelsen,gerson.cavalheiro}@inf.ufpel.edu.br

Abstract. *Since the popularization of multicore architectures, concurrent/parallel programming has become a reality in the software industry. However, CS undergraduate courses, in general, introduce this topic from the second half of the formation, reducing the time available for students to master their techniques completely. This work presents an interactive tool to perceive the impacts of exploiting different parallel programming resources on the behavior and performance of an application with strong visual appeal, a fractal generator. The built tool allows visualization of the execution exploring the parallelism between nodes (MPI), intra-nodes (OpenMP), and between instructions (vector instructions), changing in real time the set of parallelism explored.*

Resumo. *A programação concorrente/paralela é uma realidade na produção de software em função da popularização das arquiteturas multicore. No entanto, as formações em nível superior, em geral, introduzem este tópico a partir da segunda metade dos cursos, reduzindo a disponibilidade de tempo para que os alunos tem para se apropriar completamente de suas técnicas. Neste trabalho é apresentada uma ferramenta interativa para percepção dos impactos da aplicação de diferentes recursos de programação paralela no comportamento e desempenho de uma aplicação com forte apelo visual, um gerador de fractais. A ferramenta construída permite visualização da execução explorando o paralelismo entre nós (MPI), intra-nós (OpenMP) e entre instruções (instruções vetoriais), alterando em tempo real o conjunto nível de paralelismo explorado.*

1. Introdução

Graças ao advento das arquiteturas multicore, somado a maior acessibilidade a recursos de processamento, a programação concorrente/paralela assumiu um papel de destaque na indústria de software [Wilhelm et al. 2018]. Portanto, programadores devem possuir habilidades neste contexto, pois passam a ser fundamentais para explorar toda a potencialidade dos novos sistemas computacionais [Sarkar et al. 2018, Velásquez et al. 2020].

*O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

†Bolsista PIBITI CNPq.

DOI: 10.5753/sbie.2022.225734

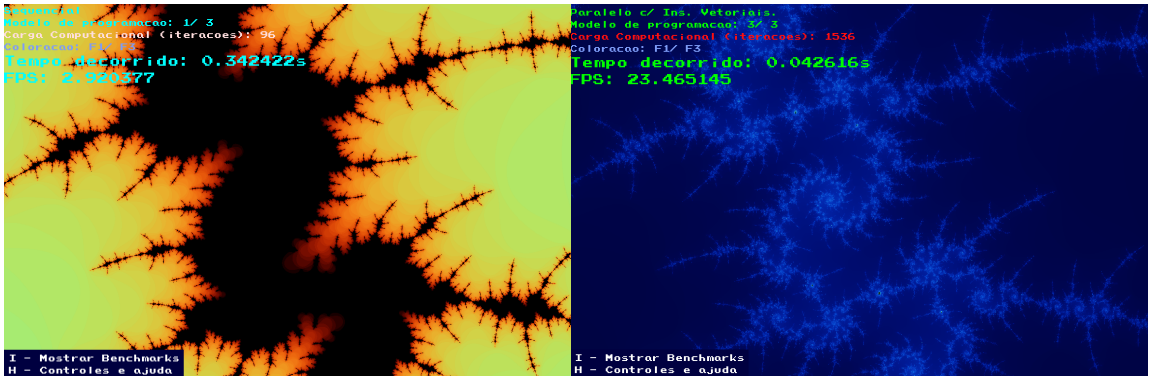

Figura 1. Nas imagens é possível ver duas regiões do fractal de Mandelbrot e a interface que permite controlar o fator de convergência, os recursos de execução e a navegação no espaço da imagem e que informa medidas de desempenho durante a execução.

Ainda assim, o ensino tradicional, nos cursos de graduação na área da computação, consideram este tópico como avançado, muitas vezes considerado como eletivo [Giacaman and Sinnen 2018].

Quanto ao momento adequado para sua introdução na grade de conhecimentos, [Vasconcelos et al. 2019] identificou que muitas discussões foram conduzidas até o momento. Não havendo um consenso, muitas vezes observa-se que estes conteúdos são ministrados na segunda metade dos cursos [Conte et al. 2020, Sitsylitsyn, Yuriy 2020]. Ainda assim, diversos trabalhos argumentam a importância de desenvolver este conteúdo nos semestres iniciais [Ko et al. 2013]. Inclusive existem diversos relatos de experimentos e práticas do ensino da programação concorrente/paralela a programadores iniciantes [Conte et al. 2020, Vasconcelos et al. 2019, Ko et al. 2013, Figueiredo and García-Peñalvo 2021, Khmelevsky and Hains 2021].

Este artigo não discute o momento em que o tópico programação concorrente/paralela deva ser inserido no currículo. O objeto é fomentar o desejo e a curiosidade sobre o tópico a partir de uma ferramenta visualmente instigante. Mais especificamente, a visualização e compreensão dos impactos na execução de um programa explorando recursos de programação em diferentes níveis, intra-nós, intra-nó e entre instruções. Os recursos de programação abordados são, respectivamente, comunicação entre processos, com MPI, multiprogramação leve, com OpenMP e exploração de instruções vetoriais, com a arquitetura AVX.

A ferramenta desenvolvida tem como diferencial oferecer um suporte visual e interativo (Figura 1). O suporte visual permite que o usuário acompanhe a execução da aplicação, tendo associada a esta visualização a informação da velocidade de execução do programa. A componente interativa permite que, dinamicamente, sejam alternados os nível de paralelismo explorados durante a execução. Como aplicação base foi utilizado o algoritmo de Mandelbrot, para geração de fractais, para instigar o aluno a interagir com a ferramenta, percorrendo as regiões do fractal e analisando impactos no desempenho pela variação dos parâmetros de execução.

A contribuição desta ferramenta é o de promover a retenção dos conhecimentos sobre a prática da programação concorrente/paralela no momento em que permite a seu

usuário perceber, a medida que interage com ela. A utilização de ferramentas como apoio ao ensino é uma prática comum ([Rios et al. 2020, Almeida Jr. et al. 2021]). Além disto, por ser oferecida na forma de código aberto, permitindo livre uso (cf. Seção 5), os estudantes tem a possibilidade de investigar o código e incluir novas alterações, oferecendo oportunidade de explorar seu computador para tirar proveito de sua autonomia em suas ações de aprendizado [Guimarães et al. 2018]. Segundo [Cave et al. 2020], mesmo programadores com alto grau de especialização no processamento paralelo, não conseguem explorar efetivamente os recursos de hardware sempre em evolução. A oportunidade de dispor de um arcabouço de software que, além de poder ser manipulado e alterado, oferece uma oportunidade lúdica de estudo, se inserindo no contexto de metodologias que oportunizam que o aluno seja protagonista no processo de aprendizado ([Calderon et al. 2021]).

O restante do artigo está organizado como segue. A Seção 2 contextualiza os aspectos relacionados a programação concorrente/paralela explorados na ferramenta apresentada. A Seção 3 apresenta, de forma breve, um algoritmo de geração de fractais, destacando os níveis de concorrência que podem ser explorados. A Seção 4 apresenta a ferramenta construída, destacando sua interface e os principais componentes que podem ser modificados e/ou estendidos com novas funcionalidades. A Seção 5 relaciona o conjunto de recursos disponibilizados publicamente e na Seção 6 é apresentado um conjunto de casos práticos da aplicação desta ferramenta. A Seção 7, por fim, conclui o trabalho, posicionando seu desenvolvimento no contexto das atividades dos autores envolvidos e apresentando perspectivas de trabalhos futuros.

2. Questões na Programação

Neste artigo, o objeto de investigação consiste em uma ferramenta de apoio a compreensão do impacto da seleção de diferentes recursos para a programação concorrente ou paralela no comportamento da execução de um programa. Esta seção caracteriza os diferentes níveis que esta concorrência/paralelismo é explorado(a) para melhor entendimento do escopo de aplicação da ferramenta proposta. Inicialmente, é apresentada a distinção entre programação concorrente e programação paralela.

Nem sempre se faz necessário apresentar a distinção entre programação concorrente e paralela, uma vez que os programas desenvolvidos neste contexto, usualmente, combinam características de ambos estilos de programação [Sadowski et al. 2011]. No entanto, estas expressões remetem à exploração de recursos distintos. Enquanto a programação concorrente visa expor propriedades da aplicação, a programação paralela relaciona-se com as características do hardware no suporte à execução. No presente texto, foi observado que o uso do termo concorrência quando discutida a aplicação utilizada e paralelismo quando discutido o programa que a implementa.

De uma forma mais específica, a programação concorrente está associada à decomposição do problema da aplicação em atividades que devam ser executadas em um programa e à especificação das sincronizações, permitindo comunicações de dados, entre estas atividades. Assim, ao descrever a concorrência de uma aplicação, o programador atenta em explicitar quais atividades podem ser executadas de forma simultânea e como devem estas reagir a mudanças de estado no programa. Embora o enfoque da programação concorrente não seja prover a implementação mais eficiente, a seleção do(s) método(s) de decomposição a ser(em) utilizado(s) depende da observação, sobretudo, das caracte-

rísticas arquiteturais sobre a qual o programa será utilizado e granularidade das tarefas – mensurada por uma relação sobre a quantidade de cálculo executada pelas tarefas e a frequência/custo das sincronizações.

A programação paralela, por sua vez, envolve a seleção de recursos de programação visando a melhor execução, em termos de obtenção de desempenho, em uma máquina específica. Cabe ressaltar que arquitetura, como multiprocessador (arquitetura com espaço de endereçamento distribuído), multicomputador (arquitetura com espaço de endereçamento distribuído ou mesmo arquitetura com funcionalidades vetoriais) é uma propriedade desta máquina. A programação paralela observa os detalhes da máquina, como número e velocidade das CPUs (ou *cores*) disponíveis, quantidade de memória e velocidade de rede, além, certamente, das suas características arquiteturais.

Em relação ao desempenho também é necessário contextualizá-lo. O desempenho de um programa concorrente/paralelo pode ser medido de diferentes formas, como tempo de execução, consumo de memória, eficiência no uso das CPUs e utilização de meios de comunicação. Não raro, uma análise do desempenho parte da observação combinada de dois ou mais destas métricas. Na ferramenta aqui apresentada, o desempenho é apresentado em termos de *velocidade* de execução, expresso em frames por segundo (FPS).

Neste trabalho, a ferramenta foi desenvolvida sobre um modelo arquitetural bastante utilizado no contexto do processamento paralelo e distribuído: um aglomerado de computadores (*cluster*) [Baker and Buyya 1999]. Tal arquitetura consiste em um conjunto de nós computacionais independentes interconectados por uma rede de comunicação, dedicados a execução de uma única aplicação por vez. É desejável, porém não imperativo, que esta rede possua grande largura de banda. Os nós, atualmente por padrão, são arquiteturas multiprocessadas por estarem dotadas de, no mínimo, 1 (um) processador multicore. Também considerado padrão, os processadores atuais contam com *cores* com recursos para processamento vetorial, nos quais instruções de baixo nível possuem a habilidade de operar de 4 ou 8 elementos de vetores em um único ciclo de máquina.

3. Níveis de Concorrência no Mandelbrot

Em muitos casos de estudo envolvendo análise de desempenho de ferramentas de programação e arquiteturas paralelas, a geração de fractais, mais especificamente o conjunto de Mandelbrot, consiste em um dos benchmarks utilizados, por exemplo em [Soto Gómez 2020, Huseinović and Ribić 2015, Al-Ali et al. 2019]. Uma das razões de sua ampla adoção reside no fato desta aplicação se caracterizar por ser embaraçosamente paralela, podendo portanto ser submetida ao estudo de casos relacionados à exploração intensiva de processamento, embora não pertencente ao cenário de aplicações para o processamento de alto desempenho com grande interesse científico [Ziogas et al. 2021]. Outra razão reside no fato de que esta aplicação gera imagens com um apelo artístico forte [Li et al. 2007]. No presente trabalho, ambas razões elegeram esta aplicação para desenvolvimento da ferramenta. Pelo apelo *artístico*, espera-se captar o interesse dos estudantes na ferramenta, motivando sua interação com esta, analisando a variação de seus diferentes parâmetros. O fato de ser *embaraçosamente paralela* permite analisar diferentes aspectos de implementação, ou seja, de introdução de paralelismo, nos diferentes níveis de concorrência apresentados ([Drakopoulos et al. 2003]).

O conjunto de Mandelbrot consiste em um fractal composto por pontos pertencen-

tes a um plano complexo. Neste plano, nos limites do espaço considerado, cada ponto é computado de forma independente. Uma sucessão de planos é definida recursivamente a partir dos planos em níveis anteriores. Considerando que $\mathcal{Z}_i$ e c representam, respectivamente, um plano i e cada um dos pontos de um plano, tem-se a Equação 1. Considerando a representação do número complexo $c = a + bi$, onde a e b representam as coordenadas no plano complexo, a Equação 2 indica como cada ponto é calculado, a partir de $\mathcal{Z}_2$, sendo representado por $(x_n + y_n i)^2 + (a + bi)$ no plano $\mathcal{Z}_{n+1}$. O nível de recursividade aplicado à sucessão de planos, em geral, é marcado nos programas na forma de um limite, considerando a precisão das computações possíveis em uma arquitetura, e o nível de resolução desejado, caso o ponto de convergência (usualmente $(x_n^2 + y_n^2) < 4$) não seja atingido.

$$\begin{aligned} \mathcal{Z}_0 &= 0 \\ \mathcal{Z}_{n+1} &= \mathcal{Z}_n^2 + c \end{aligned} \qquad (1) \qquad \begin{aligned} x_{n+1} &= x_n^2 - y_n + a \\ y_{n+1} &= 2x_n y_n + b \end{aligned} \qquad (2)$$

A imagem característica do conjunto do Mandelbrot é gerada pelo mapeamento dos números de iterações registrado no último plano complexo em uma imagem bidimensional, aplicando uma tabela de cores para representação do número complexo de cada um de seus pontos.

Conforme caracterizado nas Equações 1 e 2, o custo computacional do problema está associado os limites do plano considerado, ao número de pontos contidos neste plano. Sendo um cálculo iterativo, soma-se como critério de custo a tolerância para determinação se o valor do ponto em um plano convergiu. Como o cálculo de cada ponto é delimitado pela identificação da situação de convergência ou pelo limite de iterações, não é possível prever o custo de seu cálculo. O algoritmo obtido a partir destas equações caracteriza-se por ser embaraçosamente paralelo por permitir que o cálculo de um ponto qualquer em um plano qualquer possa ser calculado de forma concorrente ao cálculo de qualquer outro ponto em qualquer outro plano da sucessão de planos. À exceção de que o cálculo de um dado ponto c em um plano $\mathcal{Z}_i$ necessita do cálculo deste mesmo ponto no plano $\mathcal{Z}_{i-1}$. Outra manifestação de concorrência se dá no mapeamento do valor do ponto em uma cor segundo a palete de cores selecionada.

Sendo a concorrência do problema manifesta no cálculo dos pontos, o mapeamento desta concorrência sobre unidades de execução paralela considera a granulosidade da tarefa concorrente descrita no programa em termos de "quantidades de pontos por unidade de cálculo". Sendo *grande* ou *pequeno* o número de pontos, a granulosidade será dita grossa ou fina. Deve ser observado duas propriedades da classificação da granulosidade. Primeira propriedade: a relação grande e pequeno é comparativa, no contexto da aplicação, entre dois esquemas diferentes de distribuição de pontos entre tarefas. Assim, a decomposição de granularidade grossa implica na criação de um menor número de tarefas computando um maior número de pontos cada uma, que uma decomposição de granularidade fina, na qual um maior número de tarefas se comprometerá, individualmente, com o cálculo de um menor número de pontos. Segunda propriedade: a granularidade não está vinculada ao custo computacional das tarefas. Uma vez que o cálculo dos valores dos pontos do espaço de interesse é realizado por um cálculo iterativo limitado por uma verificação de convergência, o custo computacional de cada ponto difere entre si, impactando no custo computacional das tarefas criadas.

A escolha da granularidade das tarefas e do critério de convergência reflete no desempenho de execução do programa. Menos tarefas executando, individualmente, mais cálculo, menor custo operacional. Maior número de tarefas executando, cada uma com menor quantidade de cálculo, maior a probabilidade de distribuição de carga computacional entre os recursos de processamento disponíveis. A ferramenta apresentada neste trabalho é voltada ao estudo das arquiteturas multiprocessadas, dotadas de suporte à operações vetoriais, e multicomputadores. As ferramentas exploradas são baseadas em multithreading e troca de mensagens. Neste contexto, a granularidade pode ser explorada (*i*) distribuindo tarefas de grossa granularidade entre os nós de um multicomputador, (*ii*) internamente a cada nó, decompondo a tarefa em tarefas de granularidade mais fina, executadas pelas diferentes CPUs (ou *cores*) de cada nó e, (*iii*) internamente a cada CPU, explorando os recursos de processamento vetorial para então explorar a granularidade de concorrência mais fina que pode ser exposta no cálculo do conjunto de Mandelbrot.

4. Mandel2Us

A ferramenta desenvolvida recebeu o nome de trabalho de Mandel2Us. Ela é disponibilizada na forma de código aberto (Seção 6), tendo sido desenvolvida com C++ e testada nos ambientes Windows e Linux. Mandel2Us possui duas componentes: o kernel, onde a execução paralela encontra-se implementada, e a interface gráfica, no qual é realizada a interação com o usuário. O acoplamento entre estes dois componentes foi realizado de forma a permitir que o usuário visualize a renderização da imagem gerada de Mandelbrot de forma iterativa, ao mesmo tempo em que pode alterar os parâmetros de geração – incluindo a seleção dos níveis de paralelismo da arquitetura explorados. Sua percepção da formação de quadros por segundo, junto com o tempo decorrido em segundos na formação de cada quadro, mostrado na tela, dando a capacidade ao usuário analisar os diferentes níveis de desempenho oferecidos.

4.1. Kernel de execução

O kernel de execução possui uma implementação sequencial do cálculo do fractal e permite que, em tempo de execução, diferentes níveis de concorrência sejam acionados e desativados de forma a permitir observar as variações de comportamento.

O kernel de execução paralelo foi implementado utilizando MPI (Message Passing Interface), OpenMP e instruções vetoriais[1] [Schmidt et al. 2017]. Enquanto exitem diversos registros do uso de MPI e OpenMP como ferramentas ao suporte ao ensino da computação concorrente/paralela e distribuída, como [Sitsylitsyn, Yuriy 2020], observa-se que instruções vetoriais ainda são abordadas de forma introdutória em disciplinas de arquiteturas de computadores e implementação de linguagens de programação, não sendo ainda um tema recorrente em disciplinas efetivamente voltadas à programação concorrente, paralela e distribuída.

O MPI define uma interface de serviços para programação em ambientes com memória distribuída oferecendo o mecanismo de troca de mensagens para transferência de informação entre módulos de um programa executando em nós distintos de um cluster. Esta interface é aceita como padrão *de facto*, amplamente aceito nos meios acadêmico-científico e de produção, tendo diferentes implementações. A implementação utilizada é a

[1] *Vectorial instrinsics instructions*, usualmente referenciadas como instruções AVX.

OpenMPI e sua interface baseada em C++ (espaço de nomes MPI definido por OpenMPI). Caso MPI seja acionado para ser executado, a aplicação assume uma configuração centralizada, do tipo mestre/escravo. A exploração da concorrência entre nós está na decomposição da área do fractal em tarefas representando regiões que devem ser calculadas de forma independente. O paralelismo é definido pelo número de nós escravos lançados. Os nós escravos recebem do nó mestre informações sobre a região do fractal que estará sob sua responsabilidade e devolvem ao mestre o número de iterações dos pontos daquela região – realizando operações do tipo Send/Receive clássicas. O processo é contínuo até que o usuário selecione uma nova combinação de recursos de programação que não inclua MPI ou que o programa seja encerrado.

O OpenMP, por sua fez, é uma extensão à C/C++ e Fortran para programação em ambientes com espaço de endereçamento compartilhado, típico de multiprocessadores. Trata-se de um padrão definido pela indústria e amplamente aceito. A versão utilizada é aquela oferecida pelo compilador GNU G++. Oferecendo um modelo de programação multithread, OpenMP permite explorar a concorrência na execução de tarefas nas múltiplas CPUs disponíveis em um nó. Seja em uma execução isolada de um nó, seja no suporte a execução interna a um nó quando seu uso é associado ao MPI, no programa desenvolvido. A concorrência explorada neste nível está relacionada à computação dos valores associados a cada ponto do espaço complexo, realizada por dois laços paralelizados, iterando sobre os eixos x e y da região da imagem de interesse. Caso o usuário não tenha adicionado mais de uma máquina à clusterização entre nós (MPI), a execução produz continuamente a imagem correspondente à área selecionada para exibição do fractal. Caso o uso de OpenMP seja combinado com o MPI, então é gerada a fração da imagem delegada ao nó correspondente.

As instruções intrínsecas para manipulação de instruções vetoriais exploram o conjunto de instruções AVX, o qual permite a manipulação de vetores de 256-bits e, então, a operação sobre 4 valores em ponto flutuante de 64-bits (`double`) simultaneamente. Este nível de paralelismo é introduzido no laço mais interno responsável do cálculo de uma região do fractal. Assim, o laço responsável por percorrer o eixo y da imagem é incrementado em 4, sendo que cada iteração i, em uma única operação vetorial, opera o ponto na posição correspondente e os 3 elementos seguintes.

No nível de granularidade mais grossa, o conjunto de nós MPI lançados dividem a computação entre si. O número de nós é definido no lançamento do programa pelo usuário[2]. Em um nível mais fino de granularidade, os threads OpenMP possuem a responsabilidade por executar o cálculo alocado ao nó MPI. Por default, o número de threads é igual ao número de *cores* que possuir a máquina sobre a qual o nó encontra-se em execução. Existem duas implementações para o código OpenMP, uma utilizando instruções AVX, outra não. O lançamento do programa em um único nó MPI implica em uma execução sequencial, podendo o paralelismo por OpenMP e AVX ser ativado/desativado a qualquer momento. Em uma execução com n nós MPI, cada nó é responsável pelo cálculo da n-ésima parte de cada imagem de fractal, a qual é realizada, inicialmente, também em sequencial e é sujeita aos chaveamentos de paralelismo posteriores.

[2]Particularidades do lançamento do programa encontram-se no repositório da ferramenta, na documentação de instalação e uso.

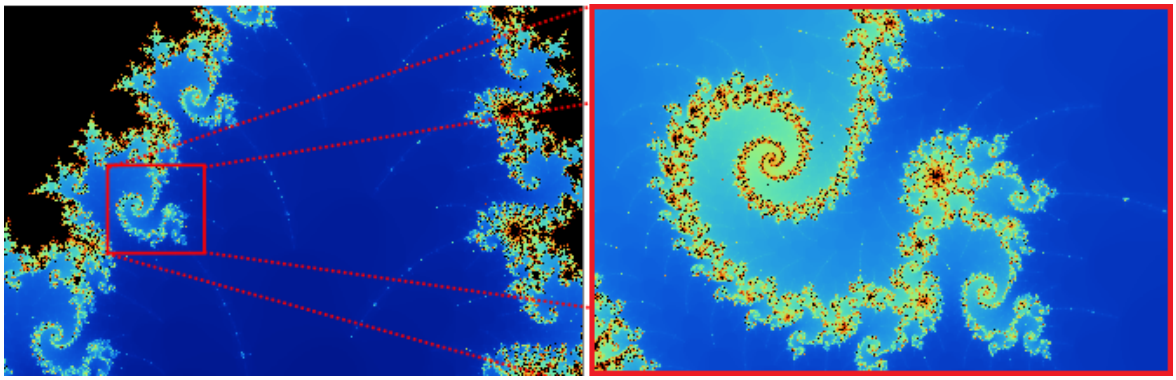

Figura 2. Ilustração do efeito de zoom.

4.2. Interface gráfica

Na Figura 1 é apresentada a interface que possibilita ajustar os parâmetros do programa em tempo-real. São apresentados dados de desempenho, qualidade de refinamento do fractal, controle da câmera e modelo de programação sendo utilizado. O objetivo é permitir que o usuário possa explorar diferentes níveis do fractal utilizando movimentações horizontais, verticais e zoom. Na Figura 2 é possível ver o efeito do zoom. O zoom é feito a partir de transformações algébricas da região de interesse para o calculo do fractal, o que permite que o usuário possa explorá-lo indefinidamente.

Permitir que o zoom seja calculado em tempo-real torna a experiência agradável. Isso possibilita que o usuário veja o efeito na qualidade e velocidade de processamento ao trocar entre modelos de programação e com isso desperte o interesse em saber mais sobre como existe tamanha diferença no desempenho.

A interface foi desenvolvida utilizando a biblioteca olcPixelGameEngine[3].

5. Material disponibilizado

A ferramenta foi disponibilizada no GitHUB em github.com/lucaszm7/Mandel2Us. No repositório também foram disponibilizados executáveis Linux e Windows, imagens de exemplo do programa e um vídeo de demonstração. O vídeo pode ser visto no seguinte link: `https://youtu.be/9SZa7MtI2_0`

O código fonte foi estruturado para ser de fácil entendimento e modificação mesmo para alunos dos semestres iniciais. Mesmo esses, devido a cuidadosa estruturação do código, são capazes de identificar no código as diferenças entre os modelos de programação e como a mesma função foi implementada nos diferentes modelos. Com isso desmistificando a programação concorrente/paralela como um tópico fora do alcance dos ingressantes nos cursos de Computação.

A partir do ciclo de exploração do código e utilização da ferramente visual é esperado que o objetivo do projeto seja alcançado: incentivar a adoção da programação concorrente/paralela. Para validar essa abordagem o seguinte estudo de caso foi desenvolvido.

[3]https://github.com/OneLoneCoder/olcPixelGameEngine

6. Estudo de caso

Esta seção apresenta um caso de estudo conduzido para validar o uso da ferramenta e a capacidade de seus usuários perceberem variações no desempenho. O procedimento de validação consistiu em enviar nas listas de alunos dos cursos de Computação da Universidade Federal de Pelotas (UFPel) um convite para participar do experimento, apresentando um formulário de avaliação e todo o material descrito na Seção 5. Os entrevistado receberam instruções para executar a aplicação nas plataformas Linux e Windows e também para gerar o executável a partir dos fontes. O formulário contou com 10 questões. Houve um total de 15 respondentes. Não houve relato de insucessos com o uso da ferramenta, tão pouco sinalizações de imprecisão nas orientações de instalação e uso. Assim, todas as tentativas de uso foram consideradas bem sucedidas. A participação como entrevistado foi voluntária, não sendo oferecida recompensa pela execução da atividade.

Nos parágrafos seguintes são apresentadas as questões desenvolvidas e a apreciação do conjunto das respostas obtidas.

(*Questão 1*) Inicialmente é solicitado para o entrevistado identificar seu semestre no curso, (*Questão 2*) se teve contato anterior com programação concorrente/paralela e, para registro, (*Questão 3*) a informação de quantos *cores* a máquina sobre a qual o experimento seria executado possui. Do total de respondentes, apenas 4 eram estudantes dos primeiros 4 semestres cursos alvo da pesquisa (26% dos entrevistados) e também 4 entrevistados informaram não ter tido contato prévio com programação concorrente/paralela. Chamou a atenção que 3 dos entrevistados que manifestaram não terem tido contato com este modelo de programação informam estar no 7 semestre ou além. Em relação ao número de *cores* disponíveis na máquina, 8 entrevistados disseram possuir computadores com 6 ou mais *cores* e nenhum indicou estar utilizando uma máquina monoprocessada.

Após orientar o entrevistado a lançar o programa em modo sequencial e executar manipulações de navegação pela imagem, (*Questão 4*) pede-se que informe se conseguiu utilizar a interface e (*Questão 5*) qual sua expectativa de ganho de desempenho ao executar em paralelo. Em relação ao grau de complexidade da interface, 75% dos entrevistados informou que não teve problemas em utilizá-la e 25% informou que teve um pouco de dificuldade. Não houve relatos de dificuldades maiores ou impossibilidade de uso. Em relação à perspetiva de ganho de desempenho, todos entrevistados afirmaram supor que haveria redução no tempo de execução, a maioria (75%) indicando que o ganho seria de 40 ou de 80%.

Os entrevistados foram então orientados a ativar OpenMP e explorar o paralelismo a este nível. Ao serem questionados (*Questão 6*) se o ganho de desempenho foi dentro do esperado, menor ou maior, 33% informou que o desempenho obtido foi o esperado, 15% que foi menor e aproximadamente 50% que foi maior.

Na questão seguinte, pediu-se que o entrevistado informasse (*Questão 7*) sua expectativa de ganho de desempenho na aplicação pela ativação do cálculo vetorial. As opções de ganho de desempenho oferecidas foram de 20, 40, 80, 160, 320 ou de mais de 640%. As respostas distribuíram-se uniformemente sobre as opções apresentadas. Após orientar de como deveria ser procedida ativação das instruções vetoriais, pediu-se que os entrevistados (*Questão 8*) manifestassem se o ganho de desempenho observado foi dentro do esperado, menor ou maior, 20% informou que o desempenho obtido foi o esperado,

40% que foi menor e outros 40% que foi maior.

Questionado de forma geral sobre os ganhos de desempenho, (*Questão 9*) foi solicitado que o entrevistado manifestasse se os resultados obtidos foram os esperados. Aproximadamente 15% dos entrevistados informou que os resultados corresponderam pouco ou muito pouco a sua expectativa. Os demais 85%, que sim ou que sim, em parte.

A questão final (*Questão 10*) buscou avaliar o interesse do entrevistado em aprofundar seus estudos na área. Em uma escala de 5 valores, todos entrevistados se posicionaram nas duas escalas de interesse mais altas.

7. Conclusão

Este artigo apresentou uma ferramenta, denominada Mandel2Us, desenvolvida para (*i*) apresentar recursos para programação concorrente/paralela e (*ii*) cativar e motivar estudantes para estudo desta área. Esta ferramenta se caracteriza por oportunizar que o usuário navegue pelo espaço de uma imagem fractal alterando diferentes parâmetros de execução. Em particular, é possível alterar o custo computacional do processamento, gerando imagens com maior ou menor definição de qualidade visual, e combinar o uso de diferentes mecanismos de suporte à concorrência/paralelismo. Executando em tempo real, o usuário pode observar o impacto no desempenho resultante da sua seleção de parâmetros. Um estudo de caso foi concebido para que estudantes de cursos de graduação em Computação utilizassem, voluntariamente, a ferramenta e anotassem sua percepção de uso. O resultado indicou que a ferramenta é plenamente funcional e suas indicações de uso adequadas, assim como, quando questionados, os estudantes se mostraram interessados em aprofundar seus estudos na área de programação concorrente e paralela.

Outro atrativo da ferramenta é ela ser disponibilizada na forma de código aberto. O aluno interessado poderá estudar o código desenvolvido e, eventualmente, inserir suas próprias alterações e, ainda, estende-la para contemplar novos recursos e ferramentas de programação.

O projeto inicial de Mandel2Us considerou que um elemento fundamental para atrair a curiosidade dos estudantes para uso de uma ferramenta para exploração de paralelismo deveria conter um elemento lúdico. Este elemento lúdico foi encontrado na geração de fractais por Mandelbrot, que além de possuir grande apelo visual ([Li et al. 2007]), é utilizado como benchmark em diversos estudos de caso do processamento paralelo ([Soto Gómez 2020, Huseinović and Ribić 2015, Al-Ali et al. 2019]).

Como contribuição, espera-se avançar no que diz respeito à formação de recursos humanos para o processamento paralelo. O desenvolvimento de sistemas contendo componentes concorrentes/paralelas é uma necessidade manifesta da indústria de software ([Wilhelm et al. 2018]). Os cursos de formação na área de Computação, no entanto, não possuem um consenso do momento em que estes conceitos devem ser introduzidos, muitas vezes se observa-se que este tópico é introduzido a partir da segunda metade do curso ([Conte et al. 2020, Sitsylitsyn, Yuriy 2020]), como é o caso nos cursos aos quais os autores deste artigo encontram-se filiados.

Com base nos resultados do estudo de caso apresentado, entendemos que a abordagem adotada foi bem recebida e resultou em manifestação de interesse na área. A partir desse resultado esperamos utilizar a ferramenta e a abordagem mencionada nas

disciplinas de programação iniciais onde somente programação serial é corriqueiramente apresentada. Esperamos como resultado de longo prazo que essa abordagem gere profissionais com conhecimentos mais amplos por terem o seu interesse despertado mais cedo na sua formação.

Referências

Al-Ali, F., Gamage, T. D., Nanayakkara, H. W., Mehdipour, F., and Ray, S. K. (2019). Supercomputer networks in the datacenter: Benchmarking the evolution of communication granularity from macroscale down to nanoscale. In *2019 29th International Telecommunication Networks and Applications Conference (ITNAC)*, pages 1–6. IEEE.

Almeida Jr., R., Farias, H., Saavedra, M., and Arauoj, J. (2021). Metodologias de ensino na programação paralela com placas gráficas: Uma revisão sistemática da literatura. In *Anais do XXXII Simpósio Brasileiro de Informática na Educação*, pages 1243–1252, Porto Alegre, RS, Brasil. SBC.

Baker, M. and Buyya, R. (1999). *Cluster Computing at a Glance*, pages 3 – 47. Prentice Hall PTR.

Calderon, I., Silva, W., and Feitosa, E. (2021). Um mapeamento sistemático da literatura sobre o uso de metodologias ativas durante o ensino de programação no Brasil. In *Anais do XXXII Simpósio Brasileiro de Informática na Educação*, pages 1152–1161, Porto Alegre, RS, Brasil. SBC.

Cave, W. C., Wassmer, R. E., Ledgard, H. F., Salisbury, A. B., Irvine, K. T., and Mulshine, M. A. (2020). A new approach to parallel processing. *IEEE Access*, 8:30287–30305.

Conte, D. J., de Souza, P. S. L., Martins, G., and Bruschi, S. M. (2020). Teaching parallel programming for beginners in Computer Science. In *2020 IEEE Frontiers in Education Conference (FIE)*, pages 1–9.

Drakopoulos, V., Mimikou, N., and Theoharis, T. (2003). An overview of parallel visualisation methods for Mandelbrot and Julia sets. *Computers Graphics*, 27(4):635–646.

Figueiredo, J. and García-Peñalvo, F. (2021). Teaching and learning tools for introductory programming in university courses. In *2021 International Symposium on Computers in Education (SIIE)*, pages 1–6.

Giacaman, N. and Sinnen, O. (2018). Preparing the software engineer for a modern multicore world. *Journal of Parallel and Distributed Computing*, 118:247–263.

Guimarães, F., Leite, M., Reinaldo, F., and Ito, G. (2018). Métodos ativos de ensino aliados com tecnologia para a prática de ensino: Um relato de experiência. In *Anais do XXIV Workshop de Informática na Escola*, pages 333–342, Porto Alegre, RS, Brasil. SBC.

Huseinović, A. and Ribić, S. (2015). Benchmark comparison of computing the Mandelbrot set in OpenCL. In *2015 23rd Telecommunications Forum Telfor (TELFOR)*, pages 994–997.

Khmelevsky, Y. and Hains, G. J. D. R. (2021). Parallel programming applied research projects for teaching parallel programming to beginner students. arXiv. arXiv:2105.13574.

Ko, Y., Burgstaller, B., and Scholz, B. (2013). Parallel from the beginning: The case for multicore programming in the Computer Science undergraduate curriculum. pages 415–420.

Li, X., Wang, Z., Che, X., and Lu, T. (2007). Artistic fractal images for complex mapping F(z) and T(z). *2007 10th IEEE International Conference on Computer-Aided Design and Computer Graphics*, pages 501–504.

Rios, M. G., Herrera, D. J. C., Lucio, K. S. N., and Vazquez, M. Y. L. (2020). AHP for a comparative study of tools used for programming learning. In Nazir, S., Ahram, T., and Karwowski, W., editors, *Advances in Human Factors in Training, Education, and Learning Sciences*, pages 356–362, Cham. Springer International Publishing.

Sadowski, C., Ball, T., Bishop, J., Burckhardt, S., Gopalakrishnan, G., Mayo, J., Musuvathi, M., Qadeer, S., and Toub, S. (2011). Practical parallel and concurrent programming. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, page 189–194, New York, NY, USA. Association for Computing Machinery.

Sarkar, V., Grossman, M., Budimlic, Z., and Imam, S. (2018). A one year retrospective on a MOOC in parallel, concurrent, and distributed programming in Java. In *2018 IEEE/ACM Workshop on Education for High-Performance Computing (EduHPC)*, pages 61–68.

Schmidt, B., Gonzalez-Dominguez, J., Hundt, C., and Schlarb, M. (2017). *Parallel Programming: Concepts and Practice*. Elsevier Science.

Sitsylitsyn, Yuriy (2020). Methods and tools for teaching parallel and distributed computing in universities: a systematic review of the literature. *SHS Web Conf.*, 75:04017.

Soto Gómez, E. (2020). Mpi vs openmp: Un caso de estudio sobre la generación del conjunto de mandelbrot. *Innovación y Software*, 1(2):12–26.

Vasconcelos, L. B. A., Soares, F. A. L., Penna, P. H. M. M., Machado, M. V., Góes, L. F. W., Martins, C. A. P. S., and Freitas, H. C. (2019). Teaching parallel programming to freshmen in an undergraduate Computer Science program. In *2019 IEEE Frontiers in Education Conference (FIE)*, pages 1–8.

Velásquez, R. A., Isaza, S., Montoya, E., García, L. G., and Gómez, J. (2020). Embedded cluster platform for a remote parallel programming lab. In *2020 IEEE Global Engineering Education Conference (EDUCON)*, pages 763–772.

Wilhelm, A., Čakarić, F., Gerndt, M., and Schuele, T. (2018). Tool-based interactive software parallelization: A case study. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '18, page 115–123, New York, NY, USA. Association for Computing Machinery.

Ziogas, A. N., Ben-Nun, T., Schneider, T., and Hoefler, T. (2021). NPBench: A benchmarking suite for high-performance numpy. In *Proceedings of the ACM International Conference on Supercomputing*, pages 63–74.