Acelerando *ray tracing* com OpenMP para WebAssembly

Bento Borges Schirmer[1], Maria Clara Bohn Silva[1], Luis Henrique Siqueri Dias[1], Thierry Weissheimer Monteiro[1], Andrea Schwertner Charão[2]

[1]Centro de Tecnologia – Universidade Federal de Santa Maria (UFSM)
97105-340 – Santa Maria – RS – Brasil

[2]Departamento de Linguagens e Sistemas de Computação – UFSM
97105-340 – Santa Maria – RS – Brasil

{bbschirmer,mcsilva,lhdias,twmonteiro,andrea}@inf.ufsm.br

Resumo. *É imperativo que o paralelismo não deixe de existir em jogos eletrônicos e ferramentas profissionais digitais que executam em navegadores de Internet. Para que um programa gráfico e interativo de* ray tracing *escrito em C execute na Web, o mesmo foi compilado para WebAssembly utilizando Emscripten, e utilizando código* open source *da Tencent, foi produzida uma versão paralelizada com OpenMP que também compila para a Web. Um experimento mostrou que, ainda que o OpenMP funcione na Web e seja observado alguma aceleração, esta esteve longe do limiar ideal. É necessário ajustes no programa e na metodologia para resultados mais significativos.*

1. Introdução

Duas classes de programas que usufruem de paralelismo são jogos eletrônicos e ferramentas profissionais digitais. Em particular, jogos realizam trabalhos pesados como: inteligência artificial de agentes, carregamento da próxima fase, simulação da física, *culling* e pré-processamentos de recursos gráficos. Ainda que haja várias formas de paralelizar programas em sistemas tradicionais, as opções são menores e limitadas para os que executam em navegadores de Internet. Essas opções são discutidas na Seção 2.

Emscripten (2015a, tradução nossa) permite "compilar código C e C++, ou qualquer outra linguagem que use LLVM, em WebAssembly, e executá-lo na Web, Node.js ou outras *runtimes* de Wasm". Outra tecnologia para C e C++, e também Fortran, é o OpenMP, que em situações simples, pode paralelizar um programa para usar vários núcleos de um processador através de uma diretiva `#pragma omp parallel for`.

OpenMP não funciona de imediato no Emscripten, mas há progresso: Emscripten (2015a, tradução nossa) "usa Clang e LLVM para compilar para WebAssembly", e o "Clang implementa inteiramente o OpenMP 4.5, quase todo o 5.0 e a maioria do 5.1/2" [THE CLANG TEAM, 2025, tradução nossa]. Assim, código compilado pelo Emscripten com opção `-fopenmp` transforma o código adequadamente através do LLVM OpenMP (2025, tradução nossa), que "quebra seções de código que devem executar em paralelo em funções separadas, que podem então ser chamadas em várias *threads*". Porém, falta uma *runtime*, mas nihui (2021, tradução nossa) informa que "no projeto ncnn, nós

implementamos uma *runtime* de OpenMP mínima para o alvo WebAssembly", sendo `simpleomp.cpp`[1] o principal código-fonte.

Dado um programa de *ray tracing*, que é fácil de implementar e trivialmente paralelizável, é verificado se o mesmo compilado para WebAssembly com Emscripten consegue ser acelerado com OpenMP tanto quanto se compilado para *desktop* nativo. Nativo é referido no presente trabalho como código de máquina do processador real, sem ser uma representação intermediária como WebAssembly, *bytecode* do Java ou linguagem intermediária do .NET.

2. Trabalhos relacionados

É relevante para a discussão APIs de paralelismo ou computação concorrente presentes em bibliotecas voltadas para a criação de jogos ou aplicativos gráficos e interativos. Rust, C, C++, JavaScript e Godot estão aptos a multiprogramação na Web, mas libGDX, Unity e Flutter não. APIs simplificadas como OpenMP hão de se marcar presença na Web.

A biblioteca de jogos libGDX (2025, tradução nossa) transforma Java em JavaScript utilizando Google Web Toolkit para produzir versões Web, mas informa que "*multithreading* não é implementado".

O motor de jogos Unity (2025, tradução nossa) adverte que, "a plataforma Web ainda não implementa *multithreading* do C# devido a limitações do WebAssembly", citando "falta de sinalização preemptiva de *threads*" e que não há "acesso direto à pilha nativa do WebAssembly. Isso afeta a coleta de lixo *multithread*".

Quanto ao motor de jogos Godot, "se a funcionalidade de *threads* estiver ativada, o projeto exportado pode usar *multithreading* para melhorar o desempenho" [LINIETSKY; MANZUR; GODOT COMMUNITY, 202-a, tradução nossa]. Godot oferece *threads* simples e duas primitivas de sincronização: *mutexes* e semáforos [LINIETSKY; MANZUR; GODOT COMMUNITY, 202-b].

Para bibliotecas de JavaScript, como Three.js, Phaser, Babylon.js, React, Vue.js e Angular, programas podem explorar Web Workers do JavaScript para fins de concorrência. Quanto ao Flutter (2024, tradução nossa), "Concorrência via *isolates* no Dart atualmente não funciona no Flutter web".

Salvo alguns cuidados, Rust consegue produzir WebAssembly *multithread* [GUIDE, 202-], possibilitando que bibliotecas de concorrência ou paralelismo sejam adaptadas para a Web, como Rayon [RREVERSER, 2025], que de acordo com Crates.io (2025), é dependência de 3994 outras bibliotecas.

Finalmente, Emscripten (2015b, tradução nossa) implementa POSIX *threads*, que "é considerado estável". Com isso, execução concorrente é viável em bibliotecas para C e C++ que funcionam na Web, como SDL, Raylib, GLFW, Sokol e Qt.

3. Metodologia

O ambiente do experimento é um notebook com um processador Intel i5-8250U que apresenta quatro núcleos, pois o Hyper Threading está desligado, e a tomada foi mantida ligada para atingir frequência de 3,40 GHz, Turbo Boost ativado.

[1] https://github.com/Tencent/ncnn/blob/master/src/simpleomp.cpp

O código-fonte do programa testado é disponibilizado publicamente.[2] Trata-se exatamente do resultado do tutorial Ray Tracing in One Weekend, de Shirley, Black e Hollasch (2024), transformado então em um programa gráfico e interativo usando SDL, que é "uma biblioteca multiplataforma projetada para dar acesso de baixo nível a áudio, teclado, mouse, joystick e hardware gráfico" [SDL, 2025, tradução nossa].

O programa testado executa ao longo de quadros, que são momentos regulares em que se lê interações do mouse e teclado, se realiza algum trabalho e finalmente se mostra algo na tela. A função de *ray tracing* é chamada em um laço de repetição paralelizado com OpenMP, pintando 100 pixels por vez, até que se passe $33,\overline{3}$ milissegundos em um quadro para obter uma taxa de atualização menor que 30 quadros por segundo.

A cena final do tutorial de Shirley, Black e Hollasch (2024) contém 487 esferas, cada pixel amostrado 500 vezes e profundidade máxima de 50 para a recursão que simula a trajetória dos raios de luz. Uma corrotina pinta 20 vezes essa cena em uma tela cheia de 1366x768 pixels usando a versão não paralela, depois 20 vezes usando quatro núcleos para a versão paralela, e então imprime o tempo médio em milissegundos renderizando do início ao fim, inclusive contando as partes sequenciais não relacionadas. Foram coletados tempos da versão "nativa", compilada para Windows usando Clang com as opções `-O3 -flto`, e da versão Web, compilada usando Emscripten com as opções `-Oz -flto` e com a *runtime* OpenMP do ncnn vinculada.

4. Resultados

Conforme a Tabela 1, o paralelismo acelerou a versão nativa 2,07 vezes, e 1,84 vezes a versão Web. Uma observação secundária é que, ainda de acordo a Tabela 1, a versão nativa sem paralelismo é 1,70 vezes mais rápida que a versão Web sem paralelismo, e 1,92 vezes mais rápida quando ambas executam em paralelo.

Tabela 1. Tempo médio, em segundos, renderizando cena com *ray tracing*

	Versão nativa	Versão Web
Sem paralelismo	2019,721	3443,640
OpenMP com quatro núcleos	975,159	1871,218

5. Conclusão e trabalhos futuros

A paralelização do programa de *ray tracing* testado foi subótima, pois se esperava uma aceleração próxima do limiar ideal — de quatro vezes para quatro núcleos. Também não foi considerado o impacto da sincronização vertical no programa. Duas alternativas são: 1) realizar uma rodada de processamento paralelo por quadro, porém fazendo cada *thread* ter seu próprio cronômetro para pausarem por conta própria; e 2) fazer o paralelismo executar durante todo o tempo, obtendo atomicamente em cada quadro intervalos da imagem já pintados para mostrar na tela.

O maior resultado foi averiguar a viabilidade do OpenMP na Web. Nesse sentido, o principal trabalho futuro é ou estender a *runtime* do ncnn para que mais recursos do OpenMP funcionem, ou adaptar a *runtime* existente do LLVM, finalmente embutindo alguma das duas no Emscripten.

[2] https://github.com/bottle2/raytracing

Referências

CRATES.IO. 2025. **rayon:** Simple work-stealing parallelism for Rust [seção sobre dependentes]. [*online*]. Disponível em: <https://crates.io/crates/rayon/reverse_dependencies>. Acesso em: 30 jan. 2025.

THE CLANG TEAM. 2025. **OpenMP Support**. [*online*]. Disponível em: <https://clang.llvm.org/docs/OpenMPSupport.html>. Acesso em: 23 jan. 2025.

EMSCRIPTEN CONTRIBUTORS. 2015a. **About Emscripten**. [*online*]. Disponível em: <https://emscripten.org/docs/introducing_emscripten/about_emscripten.html>. Acesso em: 17 jan. 2025.

_____. 2015b. **Pthreads support**. [*online*]. Disponível em: <https://emscripten.org/docs/porting/pthreads.html>. Acesso em: 17 jan. 2025.

FLUTTER. 2024. **Web FAQ**. [*online*]. Disponível em: <https://docs.flutter.dev/platform-integration/web/faq>. Acesso em: 29 jan. 2025.

GUIDE, The `wasm-bindgen`. 202-. **Parallel Raytracing**. [*online*]. Disponível em: <https://rustwasm.github.io/docs/wasm-bindgen/examples/raytrace.html>. Acesso em: 30 jan. 2025.

LINIETSKY, Juan; MANZUR, Ariel; GODOT COMMUNITY. 202-b. **Exporting for the Web**. [*online*]. Disponível em: <https://docs.godotengine.org/en/stable/tutorials/export/exporting_for_web.html>. Acesso em: 29 jan. 2025.

_____. 202-b. **Using multiple threads**. [*online*]. Disponível em: <https://docs.godotengine.org/en/stable/tutorials/performance/using_multiple_threads.html>. Acesso em: 29 jan. 2025.

LIBGDX. 2025. **HTML5 Backend and GWT Specifics**. [*online*]. Disponível em: <https://libgdx.com/wiki/html5-backend-and-gwt-specifics>. Acesso em: 29 jan. 2025.

LLVM OPENMP. 2025. **LLVM OpenMP* Runtime Library Interface**. [*online*]. Disponível em: <https://openmp.llvm.org/doxygen/>. Acesso em: 24 jan. 2025.

NIHUI. Comentário na *issue* #13892 de FLATMAX (Flax, Matt) no projeto Emscripten. **Enabling openmp**. 8 mai. 2021. Github: @nihui. Disponível em: <https://github.com/emscripten-core/emscripten/issues/13892#issuecomment-835229727>. Acesso em: 17 jan. 2025.

RREVERSER (Ingvar Stepanyan). 2025. **wasm-bindgen-rayon**. [*online*]. Disponível em: <https://github.com/RReverser/wasm-bindgen-rayon>. Acesso em: 30 jan. 2025.

SDL. 2025. **About SDL**. [*online*]. Disponível em: <https://libsdl.org/>. Acesso em: 18 jan. 2025.

SHIRLEY, Peter; BLACK, Trevor David; HOLLASCH, Steve. **Ray Tracing in One Weekend**. Versão 4.0.1. [s.l.]: [s.e.], 2024. [*online*]. Disponível em: <https://raytracing.github.io/books/RayTracingInOneWeekend.html>. Acesso em: 18 jan. 2025.

UNITY TECHNOLOGIES. 2025. **Technical limitations**. [*online*]. Disponível em: <https://docs.unity3d.com/Manual/webgl-technical-overview.html>. Acesso em: 29 jan. 2025.