Documento de Diseño

μwgpu

Ignacio Vargas Campos

Escuela de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica
II Semestre 2024

Índice

1. Detalles del Documento **1**
1.1. Fecha de la Versión y Estatus . . . 1
1.2. Organización . . . 1
1.3. Autor . . . 1
1.4. Historia de Cambios . . . 1

2. Introducción **2**
2.1. Propósito . . . 2
2.2. Alcance . . . 2
2.3. Contexto . . . 2
2.4. Resumen . . . 2
2.5. Interesados . . . 2

3. Referencias **3**

4. Glosario **3**

5. Perspectivas de diseño **3**
5.1. Contexto . . . 3
5.2. Composición . . . 3
5.3. Lógica . . . 5
5.4. Dependencia . . . 5
5.5. Información (Datos persistentes) . . . 5
5.6. Uso de patrones . . . 5
5.7. Interfaces . . . 6
5.8. Interfaz de usuario . . . 6
5.9. Estructura . . . 6
5.10. Interacción . . . 6
5.11. Dinámica de estados . . . 6
5.12. Algoritmos . . . 6
5.13. Recursos . . . 6

6. Apéndice - Alternativas de diseño **6**
6.1. Definición de recursos (“bindings”) para microbenchmarks . . . 6
6.1.1. Utilización de analizador sintáctico *naga* . . . 6
6.1.2. Permitir múltiples “bind groups” . . . 7

1. Detalles del Documento

1.1. Fecha de la Versión y Estatus

- Fecha: 6 de septiembre de 2024
- Estatus: Versión 1.0

1.2. Organización

El proyecto será llevado a cabo como parte de un proyecto de investigación en el Instituto Tecnológico de Costa Rica.

1.3. Autor

Ignacio Vargas Campos

1.4. Historia de Cambios

Versión	Fecha	Descripción de Cambios
1.0	6 de septiembre de 2024	Versión inicial del documento

2. Introducción

2.1. Propósito

El objetivo principal de este software es proporcionar un conjunto de microbenchmarks que permitan medir características específicas del rendimiento de GPUs en plataformas compatibles con WebGPU y Vulkan.

Además, el software facilita a desarrolladores la creación y ejecución de sus propios microbenchmarks personalizados a través de una biblioteca.

Esta herramienta responde a la necesidad de obtener datos de rendimiento detallados y comparativos a través de diversas combinaciones de hardware y plataformas, permitiendo a desarrolladores e investigadores optimizar sus aplicaciones para diferentes escenarios y evaluar el rendimiento general de las GPUs.

2.2. Alcance

El software, denominado "µwgpu," abarca el diseño, desarrollo e implementación de microbenchmarks para evaluar las características de hardware de GPUs, así como las interfaces necesarias para su ejecución y visualización de resultados.

El proyecto tiene dos objetivos centrales:

- Simplificar la creación de microbenchmarks para GPU de manera multi-plataforma.
- Proveer un banco de microbenchmarks comparativos que incluyan operaciones comunes para evaluar las capacidades del hardware en ejecución.

El software ofrecerá dos tipos de interfaces de usuario:

1. **Interfaz web:** Permite ejecutar microbenchmarks directamente desde un navegador, recolectar datos y mostrar resultados.
2. **Interfaz de línea de comandos (CLI):** Permite ejecutar microbenchmarks de manera nativa, compilando para Vulkan y proporcionando los resultados en formato de texto.

2.3. Contexto

El software "µwgpu" se desarrollará como parte de un proyecto de investigación del Instituto Tecnológico de Costa Rica, orientado a abordar las necesidades de la comunidad de desarrollo de software gráfico, tanto en el ámbito académico como en la industria. Este proyecto surge del interés de proporcionar herramientas útiles para el análisis y optimización del rendimiento de GPUs, especialmente en entornos que utilizan tecnologías emergentes como WebGPU y Vulkan.

El producto será utilizado principalmente por desarrolladores e investigadores que buscan recolectar y analizar datos de rendimiento detallados a través de diferentes configuraciones de hardware y software. Su objetivo es facilitar el estudio de las capacidades de las GPUs, así como la creación de programas y algoritmos que puedan ejecutarse de manera eficiente en una amplia variedad de dispositivos y plataformas. Además, esta herramienta también resultará valiosa para empresas tecnológicas que buscan optimizar sus productos y para fabricantes de hardware que desean evaluar el rendimiento de sus dispositivos en diferentes escenarios.

2.4. Resumen

- Resumen general del contenido del documento, detallando el diseño.

2.5. Interesados

- **Desarrolladores e Ingenieros en GPU**: Utilizan este proyecto para optimizar el rendimiento de aplicaciones mediante microbenchmarks específicos de WebGPU. Buscan herramientas precisas para identificar puntos críticos de rendimiento y mejorar la eficiencia.
- **Investigadores Académicos**: Emplean los microbenchmarks en estudios y experimentos relacionados con el rendimiento de GPUs, buscando datos exactos que puedan validar sus teorías y apoyar la publicación de resultados.
- **Empresas de Tecnología y Desarrollo de Software**: Implementan los microbenchmarks para optimizar el rendimiento y la experiencia de usuario en sus productos, identificando y solucionando problemas antes del lanzamiento.
- **Proveedores de Hardware**: Evalúan el rendimiento de sus GPUs en diversos escenarios para ajustar y mejorar sus productos, basándose en los resultados proporcionados por los microbenchmarks.

- **Usuarios Finales (Desarrolladores y Usuarios de Aplicaciones)**: Se benefician indirectamente de las mejoras en el rendimiento de las aplicaciones, lo cual afecta positivamente la calidad de su experiencia.

3. Referencias

- Proyecto wgpu
- Estándar de WebGPU
- Proyecto µVkCompute
- Dissecting GPU Memory Hierarchy through Microbenchmarking
- Demystifying GPU Microarchitecture through Microbenchmarking
- Nvidia CUDA Programming Guide
- GPU Atomic Performance Modeling with Microbenchmarks

4. Glosario

- **Microbenchmark**: Prueba destinada a medir el rendimiento de una característica específica de bajo nivel. A diferencia de un "benchmark" general, un microbenchmark se centra en pruebas específicas, generalmente a nivel de componentes.
- **API**: Interfaz que permite la comunicación entre diferentes sistemas de software, como aplicaciones y hardware.
- **GPU Compute**: Uso de la GPU para realizar cálculos computacionales de propósito general.
- **GPU**: Unidad de Procesamiento Gráfico.
- **CLI**: Interfaz de Línea de Comandos.
- **WebGPU**: API gráfica y de cómputo para la web que permite a los navegadores acceder a la GPU para tareas intensivas como gráficos 3D y procesamiento paralelo (GPU Compute).
- **Vulkan**: API de bajo nivel para gráficos y cómputo que ofrece un control del hardware de la GPU de manera nativa. WebGPU puede que opere a través de Vulkan.
- **wgpu**: Biblioteca de Rust que proporciona una interfaz segura y multiplataforma para trabajar con la API WebGPU y otras APIs gráficas.
- **Pipeline**: Secuencia de etapas que procesan datos en una GPU, utilizadas en gráficos o cómputo.
- **Shader**: Programa ejecutado en la GPU para realizar cálculos específicos, como efectos gráficos o cómputo general.
- **naga**: Analizador sintáctico para shaders en el ecosistema WebGPU que automatiza la gestión de recursos y compatibilidad de shaders.
- **Bindings**: Conexiones que asignan recursos de la aplicación, como buffers y texturas, a las variables utilizadas en un shader.
- **Bind Groups**: Colecciones de recursos agrupadas y enlazadas a la GPU en una única operación, optimizando el uso de shaders.

5. Perspectivas de diseño

5.1. Contexto

El diagrama de la figura 1 muestra los casos de uso que provee el sistema en el lenguaje de diagrama de casos de uso UML. Muestra los servicios proveídos por el software y su usuario principal.

El diagrama es bastante simple, dentro del "sistema" de µwgpu se tienen las funciones que este paquete de software provee a un desarrollador o investigador.

El sistema no requiere de más actores que del usuario directo que utilice el software, el cual la mayoría del tiempo será un desarrollador.

También se muestra como un servidor será el encargado de recolectar los datos de rendimiento históricos y proporcionarlos a usuarios.

5.2. Composición

El diagrama de la figura 2 muestra, utilizando el lenguaje de diagrama de componentes UML, la composición de los subsistemas de µwgpu:

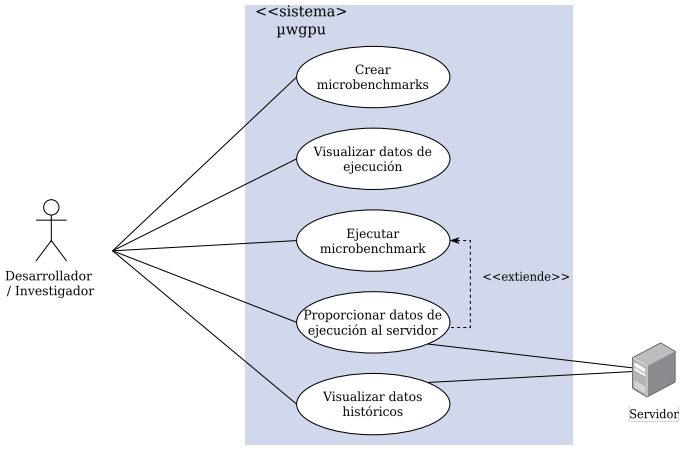

Figura 1: Casos de uso de µwgpu.

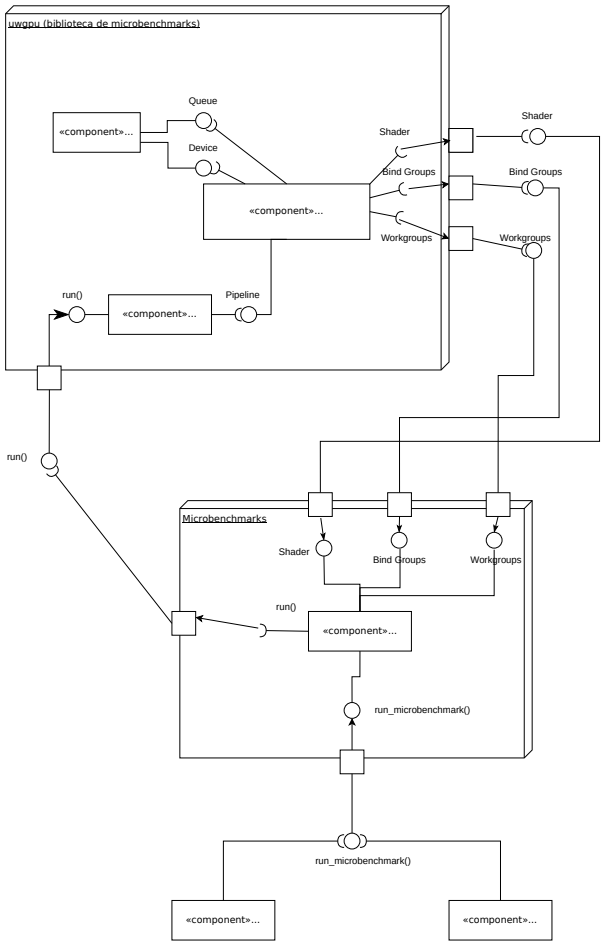

Figura 2: Diagrama de composición de µwgpu.

- La biblioteca principal “uwgpu” que se utilizar para crear microbenchmarks. Sus dependencias que cada microbenchmark debe proveer de acuerdo a lo que realice y recursos que utilice.
- La biblioteca de “microbenchmarks”. Nótese que este bloque solo se muestra 1 componente llamado “microbenchmark”. En realidad, van a haber muchos microbenchmarks con nombres más específicos, pero sus interfaces todas van a tener la misma forma: todos proveen los mismos recursos a uwgpu y las mismas interfaces a las interfaces de usuario. Por lo tanto se simplificó el bloque a mostrar solo 1 “microbenchmark” de ejemplo.
- Las interfaces de usuario que interactúan con la biblioteca proveedora de microbenchmarks para ejecutarlos y obtener sus resultados.

5.3. Lógica

El diagrama de la figura 3 muestra las clases de la biblioteca para crear microbenchmarks “uwgpu”. Es un diagrama de clases de UML. Muestra las clases expuestar ppor esta API y cómo se componen. Aquellos tipos que no se especifican en el diagrama es porque son propios del lenguaje Rust o propios del API gráfico wgpu.

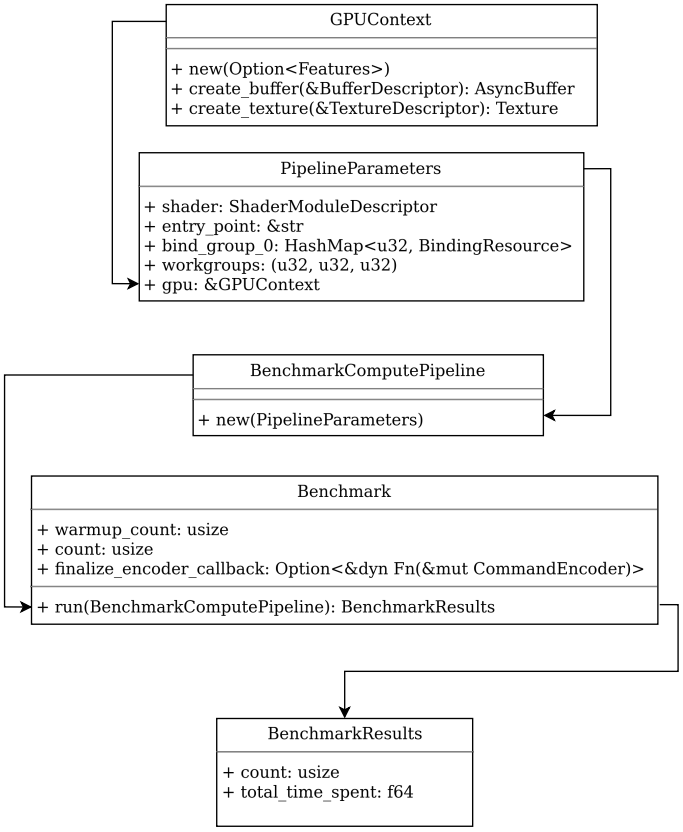

Figura 3: Diagrama de clases de biblioteca para crear microbenchmarks uwgpu.

5.4. Dependencia

No aplica.

5.5. Información (Datos persistentes)

Por definir. Se contará con un servidor que guarda información histórica de rendimiento para las ejecuciones de microbenchmarks en la página web, se documentará el formato de dicha información una vez que se diseñen los microbenchmarks y exáctamente cuáles datos son de más relevancia para cada uno.

5.6. Uso de patrones

No aplica.

5.7. Interfaces

El único servicio proveído por el software será el servidor que recolecta y provee los datos históricos. Su interfaz será expuesta a través de métodos sencillos de HTTP mostrados en el diagrama de la figura 4, el cual utiliza el lenguaje de components UML.

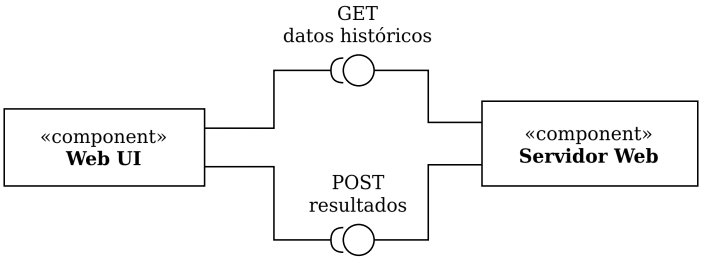

Figura 4: Diagrama de clases de biblioteca para crear microbenchmarks uwgpu.

Este diagrama también muestra cómo la interfaz de usuario web requerirá y utilizará la interfaz expuesta por el servidor.

5.8. Interfaz de usuario

Por definir. Esta sección abarcará el diseño de las interfaces de usuario CLI y la web.

5.9. Estructura

No aplica

5.10. Interacción

No aplica.

5.11. Dinámica de estados

No aplica.

5.12. Algoritmos

Se documentará posteriormente. Esta sección incluirá en detalle el diseño de los microbenchmarks creados.

5.13. Recursos

No aplica.

6. Apéndice - Alternativas de diseño

6.1. Definición de recursos (“bindings”) para microbenchmarks

En esta subsección se destacarán dos alternativas de diseño consideradas para definir cómo la biblioteca “uwgpu” obtiene los recursos necesarios para ejecutar un microbenchmark en la GPU.

6.1.1. Utilización de analizador sintáctico *naga*

La alternativa de utilizar el analizador sintáctico “naga” implicaba emplear esta herramienta para analizar los shaders y determinar automáticamente los recursos, como buffers y texturas, que se utilizan en ellos. De esta forma, la biblioteca “uwgpu” podría generar automáticamente los “bindings” y los grupos necesarios para la ejecución del shader en la GPU.

Cumplimiento de Requerimientos Esta alternativa cumple parcialmente con los requerimientos de permitir una fácil creación de microbenchmarks, ya que la automatización de la creación de "bindings" podría simplificar el proceso para desarrolladores menos familiarizados con los detalles de GPU. Sin embargo, se identificaron varios obstáculos técnicos que complicarían su implementación y afectaría el control del usuario sobre los recursos utilizados. Además, no se tenía claridad sobre la viabilidad técnica del análisis requerido para determinar automáticamente el tamaño de los buffers y texturas, lo cual conllevaría un análisis semántico complejo que podría no siempre ser preciso.

Aspectos Relacionados con la Seguridad Pública, Salud Pública, Costo Total de la Vida y Carbono Neto Cero

- Seguridad pública: no aplica.
- Salud pública: no aplica.
- Costo total de la vida: no aplica.
- Carbono neto cero: la utilización del analizador sintáctico naga podría tener implicaciones indirectas en el consumo energético. Al automatizar la creación de bindings y grupos, se requeriría mayor procesamiento y análisis de datos, lo que podría incrementar el consumo de recursos computacionales. Aunque el impacto es mínimo, en sistemas a gran escala o con ejecuciones frecuentes, la eficiencia energética podría verse afectada negativamente. Sin embargo, dado que esta alternativa no fue seleccionada, se evita un posible incremento en el uso de energía y, por ende, se contribuye marginalmente a la reducción del carbono neto.

Aspectos Relacionados con Recursos, Cultura, Sociedad y Ambiente

- Recursos: la implementación de esta alternativa requeriría recursos adicionales significativos en términos de tiempo de desarrollo y mantenimiento, debido a la complejidad del análisis automático de shaders y la integración del analizador sintáctico "naga". Además, podría generar una sobrecarga de procesamiento que aumente el consumo de recursos computacionales.
- Cultura: La automatización del proceso de "bindings" podría ser vista positivamente en una cultura orientada a la simplificación del desarrollo. Sin embargo, para los usuarios experimentados, podría generar frustración por la pérdida de control sobre configuraciones específicas que pueden ser críticas en microbenchmarks donde el detalle y el control fino son esenciales.

Justificación de la Decisión Se descartó la alternativa de utilizar el analizador sintáctico "naga" debido a la complejidad técnica y la falta de viabilidad clara en la determinación de recursos de GPU, así como la pérdida de control del usuario. Considerando estos factores, esta alternativa no es la más adecuada para cumplir con los objetivos de flexibilidad y facilidad de uso del proyecto "uwgpu".

6.1.2. Permitir múltiples "bind groups"

La segunda alternativa consistía en permitir que la biblioteca "uwgpu" soporte múltiples "bind groups" para los microbenchmarks. Esto significaría que los usuarios podrían definir y utilizar varios grupos de "bindings" para un mismo microbenchmark, ofreciendo flexibilidad en la organización y gestión de recursos.

Cumplimiento de Requerimientos Esta alternativa cumple con los requerimientos de flexibilidad para la creación de microbenchmarks más complejos que pudieran necesitar múltiples "bind groups". Sin embargo, en el contexto de microbenchmarks, donde generalmente se realizan pruebas muy específicas y de bajo nivel, esta funcionalidad adicional no aportaría un beneficio significativo para la mayoría de los casos de uso previstos.

Aspectos Relacionados con la Seguridad Pública, Salud Pública, Costo Total de la Vida y Carbono Neto Cero

- Seguridad pública: no aplica.
- Salud pública: no aplica.
- Costo total de la vida: la complejidad añadida por permitir múltiples "bind groups" podría conllevar a un aumento en los costos de desarrollo y mantenimiento del software. Los desarrolladores tendrían que invertir más tiempo en entender y trabajar con la API más compleja, lo cual podría traducirse en mayores costos de capacitación y posibles errores de implementación. Al no seleccionar esta alternativa, se evita esta complejidad adicional, optimizando el tiempo y los recursos necesarios, y potencialmente reduciendo el costo asociado al ciclo de vida del software.
- Carbono neto cero: no aplica.

Aspectos Relacionados con Recursos, Cultura, Sociedad y Ambiente

- Recursos: la implementación de esta alternativa aumentaría la complejidad de la API de la biblioteca, lo que a su vez podría aumentar el tiempo de desarrollo y mantenimiento. Además, la curva de aprendizaje para los nuevos usuarios sería más pronunciada debido a la complejidad adicional introducida por el manejo de múltiples "bind groups".
- Cultura: Esta alternativa podría ser vista de forma negativa por los desarrolladores que prefieren simplicidad y facilidad de uso en las herramientas de benchmarking. En general, la mayoría de los usuarios de "uwgpu" valoran una API más simple que permita crear microbenchmarks de manera rápida y efectiva.

Justificación de la Decisión: Se descartó la alternativa de permitir múltiples "bind groups" porque el beneficio potencial es mínimo en el contexto de microbenchmarks, mientras que la complejidad adicional haría que la API de la biblioteca sea menos intuitiva. Dado que el objetivo es facilitar la creación de microbenchmarks específicos y optimizados, se optó por mantener un enfoque más simple y directo que satisfaga las necesidades de la mayoría de los usuarios sin añadir sobrecarga innecesaria.