Diapositiva 1: Computación de Alto Rendimiento (HPC): Introducción, Retos y Avances

¿Qué es HPC?

La Computación de Alto Rendimiento (High-Performance Computing) es el uso de sistemas informáticos potentes (como supercomputadoras, clusters o redes distribuidas) para resolver problemas complejos que requieren un procesamiento intensivo de datos y cálculos. HPC permite simular fenómenos físicos, analizar grandes volúmenes de datos científicos, y ejecutar modelos matemáticos de alta complejidad en tiempos razonables.

Principales Retos:

- **Seguridad y confiabilidad**: especialmente en entornos distribuidos o de donación de recursos, donde ejecutar código externo implica riesgos (como se destaca en *Nereus* y *BOINC*).
- **Paralelización eficiente**: dificultad para adaptar algoritmos secuenciales a arquitecturas multinúcleo o distribuidas, como en simulaciones N-body o cálculo de distancias.
- **Heterogeneidad del hardware/software**: múltiples plataformas (Windows, Linux, ARM, x86) complican la portabilidad del código.
- **Gestión de datos masivos**: el volumen de información generada por experimentos como el LHC o el telescopio SKA requiere soluciones escalables para almacenamiento, transmisión y procesamiento.

Avances Clave:

- **Redes distribuidas voluntarias (Desktop Grid)**: Uso de millones de PCs personales conectados a Internet mediante plataformas como **Nereus** y **BOINC**, aprovechando la capacidad ociosa con sandbox seguros.
- **Virtualización segura en Java (JPC)**: Emulación de arquitecturas x86 para ejecutar software científico sin modificaciones en entornos heterogéneos.
- **Map-Reduce adaptado (Mycelia)**: Modelo de programación para ejecutar tareas altamente paralelas de forma tolerante a fallos.
- **Aceleración con paralelismo multinúcleo**: Optimización de algoritmos científicos con OpenMP, Numba, y threading en múltiples lenguajes (C, Python, Java).

A **por Angel Aarón Reyes Cáceres**

Diapositiva 2: Problemática — Subutilización global del poder de cómputo distribuido

Contexto computacional actual:

En física de partículas y muchas áreas científicas, el análisis de datos requiere **procesamiento intensivo**, con tareas altamente paralelizables como simulaciones Monte Carlo, análisis de eventos, y procesamiento masivo de datos.

El problema: **Los supercomputadores son costosos, limitados en disponibilidad, y poco escalables a necesidades globales**.

Oportunidad desaprovechada:

Existen más de **mil millones de PCs de escritorio conectados a Internet**, que en conjunto superan en potencia de cómputo a todos los supercomputadores combinados.

Sin embargo, solo el **0.035% de ellos están aprovechados actualmente** por proyectos como BOINC (ej. SETI@home).

Barreras técnicas:

1. **Seguridad del donante:** Código sin sandbox puede comprometer privacidad y control del sistema.
2. **Heterogeneidad del entorno:** Diferentes sistemas operativos, arquitecturas (x86, ARM), configuraciones de red.
3. **Privilegios elevados requeridos:** Muchos entornos no permiten instalar software cliente o abrir puertos.
4. **Portabilidad del software científico:** La mayoría del código está diseñado para ejecutarse como binarios nativos x86.

Diapositiva 3: Solución — Nereus + JPC: una infraestructura segura, portátil y distribuida

Propuesta técnica integral:

Nereus: Una plataforma de computación distribuida basada en Java Applets para ejecutar código **Java bytecode en sandbox**, sin necesidad de instalación.

JPC (Java PC): Un emulador completo de una arquitectura x86 escrito en Java puro, capaz de ejecutar software científico no modificado en sistemas heterogéneos.

Componentes clave:

Sandbox de seguridad de Java Applets: Restringe acceso a sistema de archivos, red y código nativo, garantizando aislamiento del código malicioso.

JPC:

- Emula la arquitectura **x86** (real mode, protected mode, periféricos, FAT32, red, reloj virtual).
- Implementa técnicas de **traducción dinámica de código (JIT)** y caching de bloques compilados.
- Soporta características como snapshots, almacenamiento remoto, y virtualización segura.

Algoritmos y marcos aplicados:

Mycelia: Un framework Map-Reduce implementado en Nereus, que permite computación distribuida tolerante a fallos.

BlackMax: Un generador de eventos de agujeros negros microscópicos del LHC, adaptado a Java y distribuido en Nereus para pruebas reales de escalabilidad.

Diapositiva 4: Resultados y Contribución a la Computación Paralela Masiva

Resultados concretos:

- **BlackMax en Nereus**: Aceleración proporcional al número de nodos; permitió escalar la simulación sin modificar el código nativo x86.
- **Mycelia**: Map-Reduce distribuido tolerante a fallos, con control de consistencia y redundancia basado en proxies HTTP.
- **JPC** demostró una emulación estable y segura con overhead aceptable frente a ejecución nativa.

Impacto en Ciencias de la Computación:

- **Portabilidad radical**: Cualquier código x86 puede ejecutarse en un navegador con Java, sin importar el sistema operativo del donante.
- **Escalabilidad global**: Topología basada en árbol con servidores Nereus permite agregar más nodos sin pérdida de rendimiento.
- **Seguridad embebida**: Todos los nodos operan en sandboxes de Java, evitando comprometer al usuario.

Contribución clave:

Esta tesis propone una **arquitectura de computación paralela distribuida segura, portátil y masiva**, habilitando el uso de **infraestructura voluntaria mundial** para problemas científicos de alto costo computacional, sin sacrificar seguridad ni interoperabilidad.

Diapositiva 5: Problemática — Ausencia de estructuras de arreglos multidimensionales en C++ estándar

Contexto del Problema:

En la computación de alto rendimiento (HPC), los *arreglos multidimensionales* son estructuras fundamentales para representar y manipular grandes volúmenes de datos científicos y de simulación numérica.

Sin embargo, **C++ estándar carece de un tipo nativo para representar vistas multidimensionales de memoria**, lo que impide:

- Integración fluida entre bibliotecas.
- Abstracción semántica coherente de operaciones matemáticas.
- Portabilidad de código entre arquitecturas heterogéneas.

Impacto Técnico:

La **falta de una abstracción de vista multidimensional** obliga a usar punteros crudos con indexación manual.

Esto introduce errores comunes, dificultad en la optimización y alta complejidad de mantenimiento.

La ausencia de un estándar también impide que herramientas como optimizadores de compiladores y analizadores de rendimiento exploten patrones semánticos comunes (ej. layouts, strides).

Requerimientos No Cubiertos:

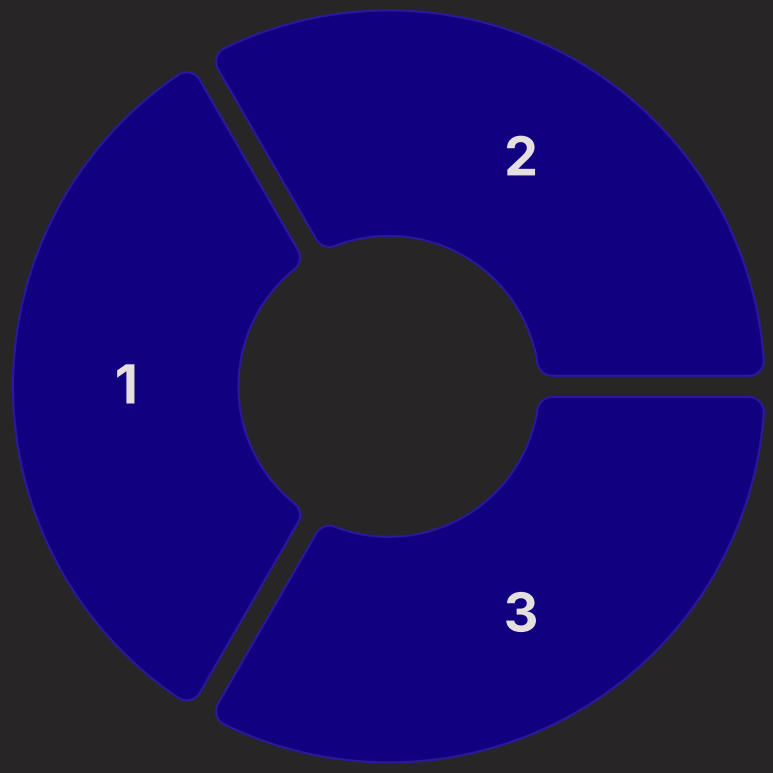
- Portabilidad de rendimiento (*performance portability*): capacidad de ejecutar eficientemente en CPUs, GPUs, y arquitecturas exascale.
- Abstracción de layouts de memoria (row-major, column-major, strided, etc.).
- Personalización de acceso a memoria para casos como acceso atómico, acceso no-aliasing, o acceso a memoria heterogénea (GPU/CPU).

Diapositiva 6: Solución Propuesta — std::mdspan como vista multidimensional estándar

¿Qué es mdspan?

Una **plantilla de clase no propietaria** (non-owning) que interpreta un bloque de memoria como un arreglo multidimensional.

Basado en el diseño del View de la biblioteca **Kokkos**, ampliamente utilizada en HPC.



Componentes Técnicos Clave:

1. **extents**: Representación de las dimensiones, combinando tamaños estáticos y dinámicos para optimizaciones en tiempo de compilación (ej. unroll loops).
2. **layout mapping**: Traduce índices multidimensionales a offsets lineales. Abstracción que soporta múltiples modelos:
   - layout_left (column-major)
   - layout_right (row-major)
   - strided y layouts simétricos.
3. **accessor**: Permite personalizar cómo se accede a la memoria:
   - atomic_ref para operaciones atómicas.
   - restrict para evitar aliasing.
   - strong pointer types para memoria heterogénea.

Ventajas del Diseño:

- Totalmente compatible con C++ moderno (desde C++11 hasta C++23).
- Separa el control de memoria (ownership) del control de acceso (*separation of concerns*).
- Facilita la integración con otras bibliotecas estándar como la futura biblioteca de álgebra lineal de C++ (P1673).

Diapositiva 7: Impacto y Validación de la Solución de mdspan

Validación Técnica:

1

- **Benchmarks** comparativos con punteros crudos muestran:
  - Casi **cero overhead** en rendimiento.
  - Algunas configuraciones incluso mejoran el rendimiento debido a *loop unrolling* y *vectorización*.
- Compatible con múltiples compiladores (GCC, Clang, ICC, NVCC) y arquitecturas (x86, ARM, POWER, CUDA).

Conclusión:

3

- std::mdspan representa un avance significativo para la **portabilidad de rendimiento en C++**, resolviendo una limitación histórica del lenguaje.
- Su diseño modular y extensible permite que HPC y aplicaciones generales converjan hacia una **abstracción común**, adaptable a arquitecturas futuras.
- Establece una base sólida para **la estandarización de algoritmos numéricos** en C++, como álgebra lineal, procesamiento de señales y simulaciones físicas.

Escenarios Críticos donde mdspan demuestra ventaja:

2

- **TinyMatrixSum**: Gran cantidad de matrices pequeñas. Uso de extents estáticos produce mejoras significativas en rendimiento (hasta 2×).
- **MatVec**: Producto matriz-vector. El cambio de layout permite explotar la arquitectura objetivo (ej. layout_right para CPUs, layout_left para GPUs).
- **Subspan3D**: Evaluación de sub-vistas complejas. Se comprueba bajo overhead incluso en usos no triviales.

Diapositiva 8: La Problemática de la Complejidad Computacional Actual

1

Retos del procesamiento de grandes volúmenes de datos en computadoras personales

En la actualidad, muchas aplicaciones —como la predicción meteorológica, simulaciones físicas o análisis de datos masivos— requieren procesar grandes volúmenes de datos con alta eficiencia.

Estos algoritmos presentan **alta complejidad algorítmica**, medida en notación **Big-O**, con órdenes de crecimiento como O(n2)O(n^2)O(n2) o O(n3)O(n^3)O(n3), que escalan mal con el tamaño de entrada.

A nivel de hardware, la mayoría de usuarios dispone de **computadoras personales con procesadores multinúcleo**, pero **no se aprovechan eficazmente sus capacidades de paralelismo**.

2

Problema específico identificado:

Existe una **brecha de conocimiento y técnica** para diseñar e implementar algoritmos de alto costo computacional optimizados para arquitecturas multinúcleo, especialmente en entornos no especializados (como PCs personales).

3

Consecuencias técnicas:

- Uso ineficiente de CPU (bajo aprovechamiento de núcleos).
- Tiempos de ejecución elevados.
- Aumento en el consumo energético y de recursos computacionales.

Diapositiva 9: Solución Propuesta — Programación Paralela Multinúcleo

Optimización de algoritmos complejos mediante procesamiento paralelo en arquitecturas multinúcleo

Enfoque de solución:

- Implementar algoritmos de alto costo computacional en lenguajes que soporten paralelismo efectivo: **Python, C y Java**.
- Aplicar técnicas de **paralelismo a nivel de hilo y vectorización**, usando herramientas como:
  - OpenMP para **C** (directivas de compilador para paralelismo automático).
  - Numba y Numpy para **Python** (JIT compilation y vectorización).
  - Threads y Runnable en **Java**.

Algoritmos seleccionados:

1. **Problema de los N-Body** (complejidad O(n2)O(n^2)O(n2)): simula las interacciones gravitatorias entre cuerpos.
2. **Matriz de Distancia Euclidiana**: cálculo intensivo de distancias entre múltiples puntos en espacio n-dimensional.

Técnicas empleadas:

- Programación multihilo y multiproceso.
- Vectorización SIMD (Single Instruction Multiple Data).
- Optimización de acceso a memoria.
- Análisis empírico de rendimiento en diferentes plataformas (PC vs HPC de Intel).

Diapositiva 10: Resultados e Impacto en el Alto Rendimiento

10x

Aceleración

Aceleraciones de hasta 10x en algoritmos complejos.

2

Lenguajes Clave

Python + Numba y C con OpenMP.

La investigación demuestra que es posible transformar equipos personales comunes en plataformas de **computación eficiente**, aprovechando técnicas modernas de programación paralela.

Resultados obtenidos:

- **Python + Numba** logró la mejor optimización en **datos de gran volumen**, reduciendo significativamente el tiempo de ejecución gracias a la compilación dinámica JIT y la paralelización transparente.
- **C con OpenMP** ofreció el mejor desempeño en **datos pequeños y medianos**, aprovechando al máximo el paralelismo a nivel de CPU.
- Comparaciones entre implementaciones secuenciales y paralelas mostraron aceleraciones de hasta **10×**, dependiendo del algoritmo y tamaño del dataset.

Evaluaciones técnicas:

- Se midieron **tiempo de ejecución**, **aceleración (speedup)** y **eficiencia**, con soporte de métricas formales de rendimiento.
- Se validó el uso de arquitecturas multinúcleo como una solución **viable y escalable** para procesamiento intensivo sin necesidad de supercomputadoras.

Impacto computacional:

La investigación demuestra que es posible transformar equipos personales comunes en plataformas de **computación eficiente**, aprovechando técnicas modernas de programación paralela.

Contribución a la disciplina:

- Promueve el uso de tecnologías accesibles de paralelismo.
- Abre la puerta a **nuevas prácticas en desarrollo de software científico** para investigadores sin acceso a infraestructura HPC especializada.