Asier Cabo Lodeiro, Martin Castro Vázquez

Arquitectura de Computadores

Grupo 47

{asier.cabo,martin.castro.vazquez}@rai.usc.es

Este informe describe la implementación y optimización global del método iterativo de Jacobi para resolver sistemas de ecuaciones lineales. Se desarrollaron cuatro versiones: una base secuencial, optimización de caché (fusión y desenrollado de bucles, blocking), vectorización SIMD con AVX256 y paralelismo OpenMP. Se evaluó el rendimiento en el Finisterrae III del CESGA en matrices de tamaño n=250, 2500 y 5000, tomando la mediana de 15 ejecuciones y comparando speedups frente a compilaciones -O0 y -O3. Los resultados muestran aceleraciones máximas de XXXXX con optimización de caché, XXXXX con SIMD y XXXXX con OpenMP. Se concluye que optimización y paralelismo explotan eficazmente la arquitectura multinúcleo y SIMD.

Programación Multinúcleo y extensiones SIMD

*Método de Jacobi, optimización de caché, speedup de rendimiento.*

1. Introducción

El método de Jacobi es un algoritmo iterativo clásico para resolver sistemas de ecuaciones lineales de la forma Ax=b, ampliamente utilizado en ingeniería y ciencias aplicadas. Aunque su sencillez lo hace atractivo, su eficiencia depende críticamente del uso óptimo de la memoria caché y de las capacidades de paralelismo modernas. Este trabajo se basa en los fundamentos teóricos de optimización de arquitecturas de memoria descritos por Hennessy y Patterson [1] y en la guía de intrinsics AVX de Intel [2] para explotar unidades SIMD y técnicas de paralelismo en memoria compartida.

El objetivo principal es implementar cuatro versiones del algoritmo de Jacobi en lenguaje C:

* Secuencial “base”, siguiendo el pseudocódigo original.
* Secuencial optimizado con técnicas de fusión y desenrollado de bucles, blocking y reordenación de accesos para mejorar la localidad de caché.
* Vectorizada con AVX256, empleando intrinsics para paralelismo a nivel de datos.
* Paralela con OpenMP, explotando múltiples hilos y distintas estrategias de reparto de carga.

Para cada versión se midió el tiempo de cómputo en matrices de tamaño n=250, 2500 y 5000, tomando la mediana de quince ejecuciones para garantizar robustez estadística. Se evaluaron los speedups relativos a compilaciones con −O0 y −O3, así como las ganancias absolutas entre implementaciones.

El resto del documento se organiza así:

Sección II describe la metodología de implementación y los detalles experimentales.

Sección III presenta los resultados obtenidos y un análisis comparativo de rendimiento.

Sección IV expone las conclusiones y propuestas de trabajo futuro.

1. Implementación y experimentación

Para abordar la evaluación comparativa de las cuatro versiones del método de Jacobi implementadas (v1–v4), esta sección se estructura en tres apartados: descripción del entorno de ejecución, detalle de la implementación y metodología de benchmarking.

* 1. Entorno de ejecución

Los experimentos se llevaron a cabo en el supercomputador FinisTerrae III del CESGA, cuyos principales parámetros se resumen en la Tabla I.

Tabla I

Valoración de especificaciones técnicas [3]

|  |  |
| --- | --- |
| Procesador | Intel Xeon Ice Lake 8352Y, 2.2 GHz |
| Núcleos | 22656 |
| Memoria | 118TB |
| SO | SUSE Linux Enterprise |
| Compilador | GCC 10.1.0 |
| Caché L1 | 49152 bytes |
| Caché L2 | 1310720 bytes |
| Línea Caché | 64 bytes |

AQUI NESTA PARTE FALO DE CALES SON OS DATOS MAIS RELEVANTES PARA NOS.

* 1. Implementación de las versiones

Para la implementación se toma como base el pseudocódigo del método Jacobi proporcionado (ver Figura 1). A partir de este, se procede a la programación de 4 variantes en C estándar, nombradas v1.c, v2.c, v3.c, v4.c, respectivamente.

La primera versión, consiste en una implementación directa del algoritmo, sin ningún tipo de optimización.

En la segunda versión, se realizan 5 pruebas para intentar implementar las siguientes mejoras:

* Reducción de instrucciones totales
* Fusión/división de lazos
* Desenrrollamiento de lazos
* Operaciones por bloques

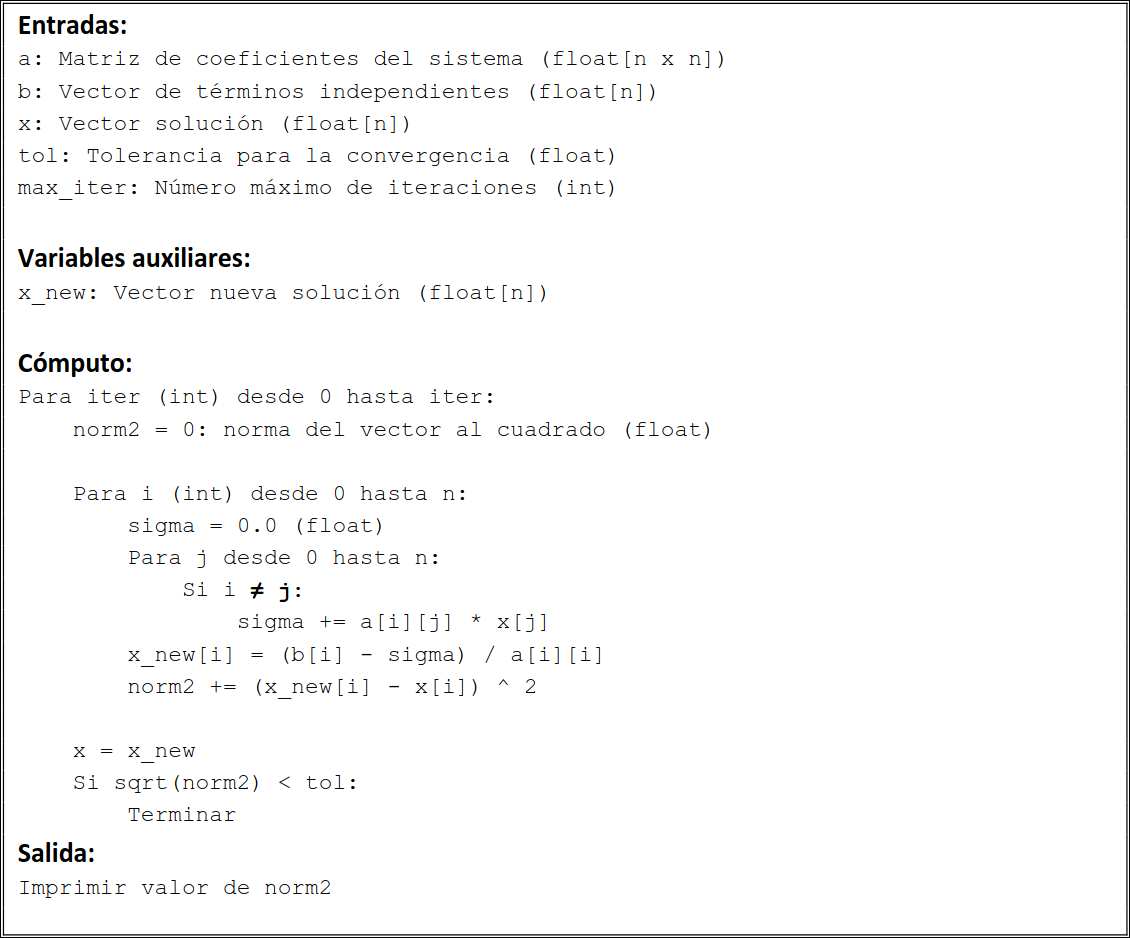
De estas optimizaciones,

CONTAR AQUI CAL ESCOLLEMOS E PORQUE.

Figura I

Pseudocódigo base del Método Jacobi

Para la tercera versión, se utiliza el procesamiento vectorial SIMD. Este es un modelo de paralelismo que permite operar varios datos de forma simultanea con una única instrucción, como se muestra en la Figura II. Es decir, en cada iteración *c[i] = a[i] + b[i];* en vez de operar únicamente con el elemento *i*, se opera con los elementos desde *i* hasta *i+x*, siendo *x* el tamaño del paralelismo, ahorrando *x* iteraciones del bucle. Para estas operaciones, es indispensable el alineamiento correcto de memoria.



Para la última versión, se usan técnicas de programación paralela en memoria compartida con OpenMP. Esta librería permite coordinar el trabajo entre varios hilos que comparten el espacio de memoria.

Figura II

Funcionamiento del paralelismo de datos

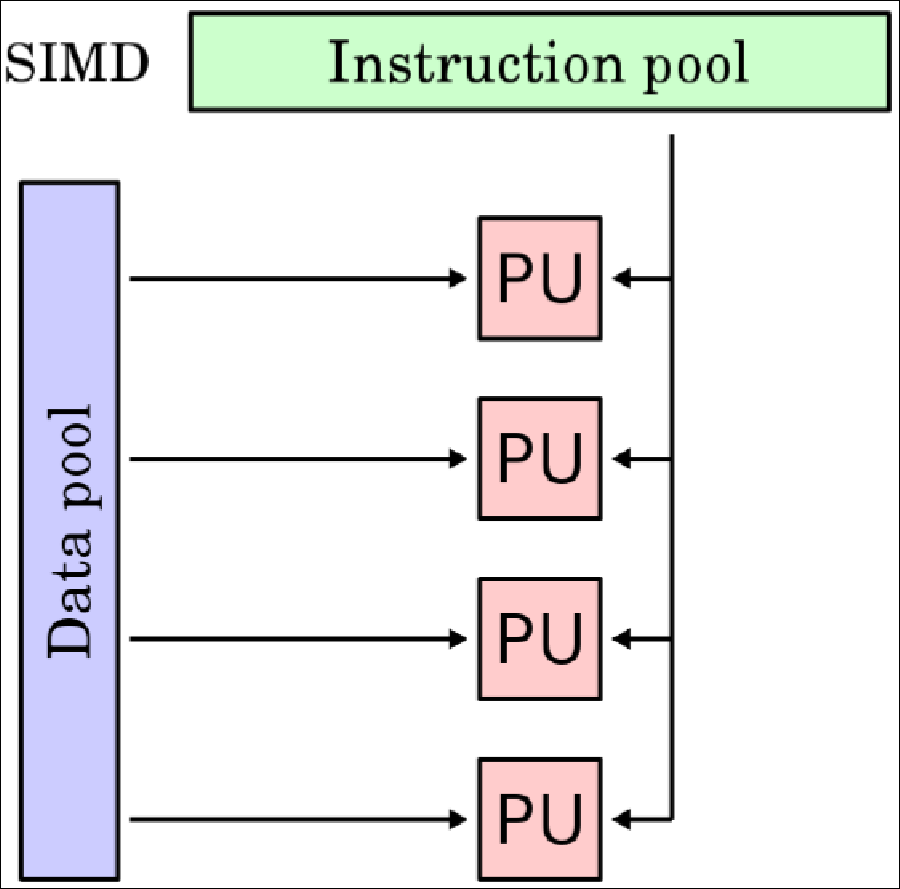


Tabla I

Valoración de especificaciones técnicas

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **TR** | **INT** | **ESC** | **DIS** | **FIA** | **TF** | **MAN** |
| 5 | 10 | 9 | 7 | 7 | 8 | 3 |
| **COM** | **SEG** | **COS** | **TAM** | **CON** | **Treal** | **IS** |
| 1 | 6 | 5 | 10 | 10 | 2 | 6 |

Si son datos que nosotros no hemos obtenido, sino que son sacados de libros, artículos o webs debemos indicar la fuente de que fueron obtenidos.

1. Resultados y análisis

Se presentan resultados en tablas o en figuras de modo que todas las tablas y figuras tengan un formato similar. Si son datos que nosotros no hemos obtenido, sino que son sacados de libros, artículos o webs debemos indicar la fuente de que fueron obtenidos. Se presentan los resultados.

En los resultados debemos describir cómo hemos realizado los experimentos para que puedan ser reproducidos por terceros. Hay que comentar los resultados antes de pasar a la siguiente sección.

Esta sección puede estar dividida en varias subsecciones.

1. Conclusiones

El apartado de conclusiones explica qué problema ha sido tratado en el documento, qué fue lo que se hizo y cómo se trabajó. Será una especie de recordatorio de todo el documento.

Se deberán ir desglosando las principales observaciones, conclusiones que podemos extraer y los problemas que se han ido encontrando al ir realizando nuestro trabajo.

Además, este apartado es el adecuado para exponer cuales podrían ser posibles trabajos futuros que completen lo explicado en este trabajo.

Se deberán ir desglosando las principales observaciones, conclusiones que podemos extraer y los problemas que se han ido encontrando al ir realizando nuestro trabajo.

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.

Referencias

1. J. L. Hennessy y D. A. Patterson, Computer Architecture: A Quantitative Approach, 5.ª ed., Morgan Kaufmann, 2012.
2. Intel, “Intel Intrinsics Guide — AVX,” 2023. <https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html> (Accedido: 15 de abril de 2025)
3. CESGA’s new FinisTerrae III. <https://www.cesga.es/en/cesga-actualiza-el-finisterrae/?utm_source=chatgpt.com> (Accedido: 4 de mayo de 2025)

<https://cesga-docs.gitlab.io/ft3-user-guide/batch_system.html>