Análisis del rendimiento

Jesús Sánchez Cuadrado

Resumen. En este documento de explica cómo realizar el análisis del rendimiento de un programa paralelo. Los objetivos de aprendizaje son:

- Entender y aplicar métricas de evaluación del rendimiento de programas paralelos.
- Cómo reportar los datos en un informe.
- Herramientas de depuración y análisis.

Índice

1. Introducción

Dado un algoritmo que se desea paralelizar, o un algoritmo que ha sido paralelizado, es importante disponer de métricas y herramientas que permitan analizar su tiempo de ejecución y su escalabilidad.

Habitualmente el objetivo es paralelizar un programa o un algoritmo para el cual disponemos de una versión secuencial, que nos sirve de referencia. La versión secuencial también es útil para poder probar el programa paralelo, esto es, comprobar que con las mismas entradas el programa paralelo se comporta igual que el secuencial. Estaremos interesados en mejorar el rendimiento del programa original, y utilizaremos métricas para determinar cuánto ha mejorado el rendimiento.

Las **métricas** permiten caracterizar, evaluar y comparar diferentes implementaciones de un programa. En este caso, la versión secuencial y paralela, pero algunas de las métricas son útiles para comparar diferentes implementaciones de un mismo programa. Para poder aplicar métricas es necesario realizar mediciones. En particular, estaremos interesados en el **tiempo de ejecución**, pero hay otras métricas que pueden considerarse, como el consumo de memoria, el gasto energético, etc.

Por otra parte, es necesario disponer de herramientas y técnicas que ayuden a razonar sobre el rendimiento y la corrección de los programas. Las herramientas que analizan el tiempo de ejecución de diferentes partes de un programa se denominan **profilers** y permiten identificar cuellos de botella (esto es, partes del programa que tienen un rendimiento pobre y afectan más negativamente al rendimiento global).

2. Métricas de rendimiento

La principal motivación de la programación paralela es conseguir mejor rendimiento. Esta mejora puede conseguirse de diversas maneras: reduciendo el tiempo de ejecución, consiguiendo tratar con problemas más grandes y complejos que no sería posible abordar con programación secuencial o consiguiendo mayor precisión en los cálculos.

Por tanto, es necesario disponer de métricas que permitan evaluar hasta qué punto se han obtenido mejoras al programar un algoritmo de manera paralela.

2.1. Speedup

La métrica básica es el *speed up* que indica la mejora del tiempo de ejecución del programa paralelo con respecto al secuencial. La siguiente fórmula muestra cómo calcularlo, donde $T_{secuencial}$ es el tiempo de ejecución del algoritmo secuencial y $T_{paralelo}$ es el tiempo de ejecución del algoritmo paralelo.

$$speedup = \frac{T_{secuencial}}{T_{paralelo}} \tag{1}$$

El valor del *speedup* estará entre 0 y p, donde p es el número de procesos o hilos con el que se ha ejecutado el programa paralelo. Idealmente, querríamos obtener un *speedup* de p lo cual normalmente no es posible. Así, el *speedup* será menor que el número de procesadores. Si no fuera así, una posible razón es que la implementación del algoritmo paralelo es mejor que la del secuencial, por lo que podría tener sentido utilizar el mismo esquema para implementar nuevo algoritmo secuencial mejor.

En algunos casos puede haber speed-up superlineal por una mejor gestión de la memoria al usar más procesadores (aunque en otras ocasiones sucede lo contrario, usar varios procesadores aumenta el número de fallos de caché).

Elegir el programa secuencial

Para realizar la comparación deberíamos utilizar el mejor algoritmo secuencial que conozcamos. Si utilizamos un mal algoritmo (por ejemplo, realizamos un bucle anidado cuando un bucle simple es suficiente) es posible que obtengamos un buen *speedup* al paralelizar, pero el resultado es engañoso. Esto es debido a la Ley de Amdahl (ver a continuación) ya que se aumenta la fracción paralela de manera artificial.

2.2. Eficiencia

El *speedup* da una noción de si existe una mejora en el rendimiento. La métrica llamada *eficiencia* indica qué porción del tiempo los procesadores están haciendo trabajo útil. Se calcula como el *speedup* dividido entre el número de procesadores que trabajan en paralelo.

$$eficiencia = \frac{speedup}{p} = \frac{T_{secuencial}}{p \cdot T_{paralelo}} \tag{2}$$

Por tanto, la eficiencia tiene un valor entre 0 y 1, y representa el porcentaje promedio del tiempo que cada procesador se está utilizando durante la ejecución paralela. Si *eficiencia = 1* significa que todos los procesadores se están utilizando completamente y corresponde a un *speedup* lineal.

2.3. Ley de Amdahl

La Ley de Amdahl establece un límite para el incremento del speedup de acuerdo a la fracción del programa secuencial que puede ser paralelizada [1]. La idea fundamental es que la mejora de un programa debida al paralelismo está limitada por la cantidad de programa que puede paralelizarse[1].

El tiempo de ejecución de un programa secuencial se puede clasificar en tres categorías:

- Tiempo dedicado a realizar trabajo que no puede paralelizarse. Es la parte inherentemente secuencial: T_{psec}
- Tiempo dedicado a realizar trabajo que sí que puede paralelizarse: T_{ppar}.
- Sobrecarga debida a la introducción del paralelismo (ej., operaciones de comunicación y cálculos redundantes).

Por tanto, podemos decir que el *speedup* siempre cumplirá que es menor qué la siguiente expresión. La idea es que si tenemos p procesadores el tiempo de ejecución del programa en paralelo ($T_{paralelo}$) estará compuesto por la parte inherentemente secuencial (T_{psec}), la parte paralelizable cuyo tiempo de ejecución se puede dividir idealmente por p (T_{ppar}) más la sobrecarga. Puesto que en muchos casos la sobrecarga es complicada de estimar y sabemos que siempre será mayor que 0, podemos eliminarla de la ecuación.

$$speedup \leq \frac{T_{psec} + T_{ppar}}{T_{psec} + T_{ppar}/p + Sobrecarga} \leq \frac{T_{psec} + T_{ppar}}{T_{psec} + T_{ppar}/p} \tag{3}$$

La fracción f del tiempo total de ejecución del programa secuencial que es no es paralelizable, es decir que es inherentemente secuencial, es $f = T_{psec}/(T_{psec} + T_{ppar})$ y por tanto manipulando la ecuación tenemos:

$$T_{psec} + T_{ppar} = T_{psec}/f$$
$$T_{ppar} = \frac{T_{psec}}{f} - T_{psec}$$

Sustituyendo estas dos definiciones en la Ecuacion 4 se puede derivar la fórmula de la Ley de Amdahl.

[1] La Ley de Amdahl se puede aplicar a cualquier técnica de mejora del rendimiento. Es cuestión de cambiar "paralelismo" por la técnica concreta que se use para mejorar el rendimiento.

Ejemplo

Dado un programa secuencial, ejecutamos diferentes *benchmarks* y se determina el 90 % del tiempo de ejecución se dedica a partes que pueden ser paralelizadas, y el 10 % en partes que no pueden serlo. ¿Cuál es el máximo speedup con 8 procesadores?

$$Speedup(n, 8) \leq \frac{1}{0{,}1 + \frac{1-0{,}1}{8}} = 4{,}7$$

El speedup up está seriamente limitado por la capacidad que tengamos identificar partes paralelizables.

$$
\begin{aligned}
speedup &\leq \frac{T_{psec} + T_{ppar}}{T_{psec} + T_{ppar}/p} \Rightarrow \\
speedup &\leq \frac{T_{psec}/f}{T_{psec} + \frac{T_{psec}/f - T_{psec}}{p}} \Rightarrow \\
speedup &\leq \frac{T_{psec}/f}{T_{psec} + T_{psec} \cdot \frac{1/f-1}{p}} \Rightarrow \\
speedup &\leq \frac{1}{f + \frac{1-f}{p}}
\end{aligned}
\tag{4}
$$

La Ley de Amdahl ofrece una cota superior del *speedup* que podemos esperar al paralelizar un programa secuencial dado. La principal consecuencia es que si solo podemos paralelizar un parte del programa, el speedup va a estar limitado. La Fig. 1 muestra la evolución del speedup según la fracción secuencial del programa.

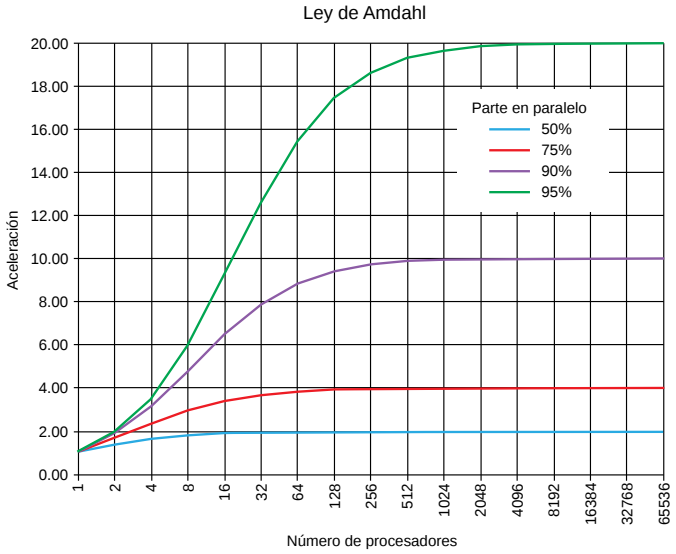

Figura 1: Evolución del speedup según la fracción secuencial. Fuente: Wikipedia

Por tanto, la eficiencia de la paralelización (el tiempo que los procesadores están realizando trabajo útil) se decrementa a medida que añadimos más procesadores. Esta es una de las razones por las que la computación paralela es más útil para abordar tipos de problemas que sean muy paralelizables.

Una limitación de Ley de Amdahl es que asume que se desea resolver un problema de tamaño fijo lo más rápido posible. Sin embargo, en muchas ocasiones el paralelismo nos permite abordar problemas de mayor tamaño. Si incrementamos el tamaño del problema, es posible que la fracción paralelizable tenga más trabajo, y por tanto podremos conseguir más speedup con los mismos procesadores[2].

2.4. Ley de Gustafson-Barsis

La Ley de Amdahl permite analizar la escalabilidad potencial de un programa secuencial y proporciona una cota superior. El principal inconveniente de la Ley

[2]Estamos asumiendo que el la sobrecarga al aumentar el tamaño del problema se incrementa menos que el tiempo de ejecución.

de Amdahl es que el problema (la carga de trabajo) no puede aumentarse para corresponderse con el poder de cómputo al aumentar el número de procesadores. En otras palabras, tamaño de la entrada impide el escalado del rendimiento.

En ocasiones estaremos simplemente interesados en reducir el tiempo ejecución. Sin embargo, existen muchas aplicaciones para las que podremos estar interesados en mejorar la precisión: si disponemos de más procesadores podemos mantener el tiempo de ejecución fijo y con ello conseguir una mayor precisión en los cálculos.

La Ley de Guftason-Barsis dice que cualquier problema suficientemente grande se puede paralelizar con un speedup (con α la porción secuencial del programa y p el número de procesadores):

$$speedup \leq p + \alpha \cdot (1 - p) \tag{5}$$

El tiempo de ejecución de un programa paralelo con p procesadores puede decomponerse en $T = a + b$, donde a es el tiempo de ejecución secuencial y b es el tiempo de ejecución en paralelo en cada procesador (es decir, $Tpar/p$). El tiempo de ejecución de ese mismo programa si lo ejecutaramos en secuencial sería: $a + p \cdot b$ (porque asumimos que todos los procesadores hacen el mismo trabajo y que el tiempo de la fracción secuencial es la del programa paralelo). Por tanto, partimos de un programa paralelo y se trata de determinar cómo de rápido es ese programa si se ejecutara en un único procesador. Esto se basa en la suposición de que la cantidad total de trabajo varía línealmente con el número de procesadores y mantenemos fijo el tiempo de ejecución. Por tanto, el speedup con respecto al programa secuencial que podemos obtener es:

$$speedup = \frac{a + p \cdot b}{a + b} \tag{6}$$

Si definimos α como la fracción secuencial del programa, $\alpha = \frac{a}{a+b}$, tenemos:

$$speedup = \alpha + p \cdot (1 - \alpha) = p - \alpha \cdot (p - 1) \tag{7}$$

La Ley de Gustafson-Barsis es aplicable cuando es posible incrementar el tamaño del problema al mismo tiempo que incrementamos el número de procesadores. Está basada en la idea de que si el tamaño de un problema puede crecer al mismo tiempo que crece p (el número de procesadores), entonces la fracción secuencial de la carga de trabajo no representará la parte dominante de dicha carga.

2.5. Estudio experimental

El estudio experimental de un algoritmo o un programa paralelo permite conocer el comportamiento de un algoritmo en la práctica, en función de diferentes entradas de datos. Así, los resultados del estudio experimental podrán compararse con la estimación teórica que se tenga.

La primera cuestión que debe abordarse es qué tipo de conclusiones se quieren obtener. Por ejemplo, podemos estar interesados en conocer el speedup a medida que aumentamos el número de procesadores (Ley de Amdahl) o si podemos mejorar la precisión incrementando la carga y aumentando el número de procesadores (speedup escalado), etc. Se deben diseñar los experimentos en función de estas cuestiones.

Para evaluar la mejora del speedup al aumentar el número de procesadores elegiremos una entrada de tamaño adecuado y se ejecutará el programa incrementando el número de hilos/procesos. Esto permitirá observar la tendencia y estudiar el comportamiento.

Para evaluar la escalabilidad podemos fijar incremental el tamaño del problema linealmente con el número de procesadores utilizado.

También hay que tener en cuenta que ejecuciones distintas, con los mismos parámetros (tamaño de entrada, número de hilos) puede dar lugar a tiempos distintos. Esto es particularmente cierto en lenguajes manejados como Java. Por tanto, puede ser importante realizar varias ejecuciones y utilizar la media o la mediana.

3. Herramientas de análisis

El desarrollo de aplicaciones de alto rendimiento, en particular, en el caso de programas paralelos requiere considerar los posibles cuellos de botella de la implementación, el impacto de los algoritmos y las estructuras de datos en el rendimiento, así como tener en cuenta que existen relaciones entre el software y el hardware que pueden hacer necesario programar ciertas partes de la aplicación de manera optimizada para ese hardware.

El proceso que se sigue para optimizar una aplicación suele ser:

1. Construir una primera versión del sistema.
2. Preparar datos de prueba que sean significativos. Por ejemplo, es importante tener entradas de diferentes tamaños para poder observar si el sistema es escalable o no.
3. Ejecutar pruebas de rendimiento. Si el rendimiento no es adecuado hay que comenzar un proceso de *tuning*:
   - Identificar cuellos de botella. Para esto se puede instrumentar el programa a mano o bien se pueden usar herramientas de análisis del rendimiento.
   - Refactorizar el programa para solucionar el cuello de botella.
   - Volver al punto 3.

3.1. Profilers

Un *profiler* se encarga de capturar eventos sobre la ejecución del programa y agregarlos. Por ejemplo, un tipo de evento podría ser el inicio de la ejecución de una función y otro evento el de finalización. El profiler captura estos dos eventos cada vez que se ejecuta la función, calcula el tiempo de ejecución cada vez y lo agrega para mostrar el tiempo total de la ejecución del programa que se ha dedicado a ejecutar esa función. Eso permite comparar con otras funciones y detectar cuáles pueden ser cuellos de botella. Otro ejemplo son los fallos de caché que un profiler podría contar y agregar.

Para obtener obtener los eventos, un profiler debe monitorizar la ejecución de un programa. Hay tres técnicas básicas:

- Monitorización a nivel de hardware. La mayoría de los microprocesadores actuales incluyen contadores a nivel de hardware (*hardware counters*) que permiten la obtención de información sobre el comportamiento del procesador al ejecutar un programa. Estos contadores son registros especiales de la CPU que realizan el conteo de eventos a nivel hardware como el número de instrucciones ejecutadas, número de fallos de caché, saltos mal predichos, etc.

  Algunas herramientas, como *perf*, utilizan estos contadores para mostrar estadísticas e información agregada.

  El principal inconveniente es que la información obtenida es poco configurable, ya que los eventos hardware son fijos y globales.

Elegir el programa secuencial

¿Y si no podemos construir un algoritmo secuencial? Si el sistema es grande o no tenemos los recursos para dedicar el tiempo a construir la versión secuencial, como aproximación podemos asumir que asimilar la ejecución con un hilo o un proceso al algoritmo secuencial.

Leaky abstractions

Las abstracciones permiten abordar la complejidad del software eliminando los detalles de las capas inferiores. Sin embargo, las abstracciones no son perfectas y en muchas ocasiones es necesario conocer los detalles de las capas inferiores (ej., el hardware) para poder hacer un uso adecuado de los recursos.
Por ejemplo, la iterar un array bidimensional, la forma en que se haga el recorrido tiene mucha importancia a la hora de evitar fallos de caché.
Ver `https://en.wikipedia.org/wiki/Leaky_abstraction`

- Muestreo (Sampling). Consiste en interrumpir la ejecución del procesador cada cierto tiempo para consultar su estado. Esto permite, por ejemplo, saber qué instrucciones se ejecutan más frecuentemente y estimar el tiempo de ejecución de las funciones. La principal ventaja es que no es intrusiva y se puede configurar aumentado la frecuencia de muestreo (aunque a costa de afectar el rendimiento). Un inconveniente es que la información ofrecida es estadística, y por tanto no es tan preciso como otros métodos.

- Instrumentación. En este caso la aplicación se modifica para insertar rutinas que capturan los eventos de interés. Hay diferentes tipos de instrumentación, según a qué nivel se realice. Se puede modificar el código fuente, bien a través de alguna herramienta de transformación de código, por el compilador, o incluso manualmente. También se puede modificar directamente el programa objeto, normalmente a través de alguna herramienta asociada al profiler. Sería posible incluso realizarlo en tiempo de ejecución (p.ej., VisualVM).

  Es una aproximación flexible pero más intrusiva que las demás.

Aspectos de rendimiento. Habitualmente cada herramienta está especializada en cierto aspecto del rendimiento (aunque hay herramientas que analizan varios aspectos).

- Tiempo de ejecución. Este es el tipo de herramienta usada con más frecuencia. Son de utilidad para detectar "hotspots", es decir, regiones de código donde el programa pasa la mayor parte del tiempo. Si se consigue optimizar este tipo de regiones el rendimiento global del programa mejorará.

- Ejecución de instrucciones. Existen herramientas que muestran información a nivel de las instrucciones ejecutadas por el procesador. Eso permite identificar si se están usando operaciones costosas (ej., divisiones), fallos en las predicciones de salto, etc.

- Patrones de acceso a memoria.

- Uso de memoria. Estas herramientas permiten analizar el uso de la memoria para detectar problemas debido a que se usa demasiada memoria o a fugas de memoria.

- Entrada/Salida. Las operaciones de entrada/salida pueden ser un cuello de botella importante en las aplicaciones, en particular en aquellas que usan gran cantidad de datos. Las herramientas de análisis pueden ofrecer información sobre este aspecto también.

- Ejecución paralela. Estas herramientas ofrecen soporte para analizar diferentes aspectos, según el modelo de programación. Para paso de mensajes (ej., MPI) se pueden analizar los patrones de comunicación, el coste de las comunicaciones, etc. Para memoria compartida se pueden analizar cuestiones como sincronización, sobrecarga, etc.

3.2. Herramientas de profiling

A continuación se presentan brevemente algunas herramientas de *profiling*. Aunque algunas no son multi-hilo, pueden resultar útiles para estudiar el rendimiento del programa secuencial.

gprof. Es una herramienta de análisis del rendimiento para Linux. Utiliza instrumentación (a través de `gcc`) y sampling. Para poder usar `gprof` hay que compilar con la opción `-pg`. Esto instrumentará el código para que al finalizar la

ejecución del programa se genere un fichero `gmon.out` que contiene las estadísticas de la ejecución del programa. El comando `gprof ./mi_programa` muestra la información sobre la ejecución.

perf. Es un profiler para Linux que accede a los eventos del sistema para ofrecer información de grano fino sober la ejecución del sistema o de un programa[3].

La forma más sencilla para evaluar un programa es:

```
$ sudo perf -d stat ./programa
# -d significa "detailed"
```

callgrind Valgrind es una herramienta de análisis que incluye una extensión para realizar profiling. La aproximación de Valgrind es crear un entorno de ejecución simulado en el que se realizan comprobaciones adicionales, como por ejemplo capturar el tiempo de ejecución de las diferentes partes del código[4][5].

Para usar Valgrind hay que compilar con la opción `-g`. Se ejecuta con:

```
$ valgrind --tool=callgrind ./programa
```

Se puede utilizar la herramienta KCacheGrind para cargar la información generada y estudiarla a través de la interfaz gráfica.

4. Escritura de informes técnicos

Un informe técnico es un documento que describe el proceso seguido y los resultados obtenidos al realizar cierta tarea de investigación o de ingeniería. Un buen informe técnico permite transmitir ideas técnicas de manera estructurada [2].

Un informe técnico se debe escribir utilizando un lenguaje técnico y formal. El objetivo es que el lector del informe pueda comprender el trabajo realizado sin necesidad de explicaciones adicionales.

¿Es legible mi informe?

El informe debe revisarse poniéndose en el papel de un lector que tiene un conocimiento básico del tema, pero que no conoce los detalles. Se debe responder a la pregunta ¿sería capaz este lector de entender los detalles de lo que se ha hecho, los resultados obtenidos e incluso de replicar el trabajo?
Otra forma de evaluar el informe es pedir a un compañero que lo revise y de su opinión acerca de si ha comprendido el contenido del documento o no.

4.1. Elementos del informe

Portada. Debe indicar el título, el autor y la fecha.

Tabla de contenidos. Es conveniente generarla automáticamente y su formato debe ser claro para que el lector pueda identificar las secciones y subsecciones de un vistazo.

Introducción. Presenta el tema del informe de manera resumida. Al terminar su lectura el lector debería tener una idea clara de qué se encontrará al leer el resto del informe. Puede estructurarse de la siguiente manera:

- Contexto.
- Motivación.
- Aproximación seguida.
- Resultados obtenidos.
- Organización del resto del documento.

[3] La página de Brendan Gregg, ingeniero de Netflix, contiene mucha información y bien organizada: `http://www.brendangregg.com/linuxperf.html`

[4] `https://docs.kde.org/trunk5/en/kdesdk/kcachegrind/kcachegrind.pdf`

[5] `https://valgrind.org/docs/manual/cl-manual.html`

Objetivos. Describe los objetivos que se pretenden conseguir con el trabajo.

Estado del arte/Trabajo relacionado. Antes (y durante) la realización del informe es importante haber estudiado trabajos sobre los que se base el trabajo realizado. También se deben incluir trabajos que sean complementarios, explicando por qué lo son.

Dependiendo del tipo de informe puede ser no ser necesario incluir esta sección. Por ejemplo, en un Trabajo Fin de Grado es una sección importante porque pone en contexto el trabajo realizado, mientras que en la práctica de una asignatura puede ser en muchas ocasiones innecesario.

Cuerpo. Dependiendo del tipo de informe esta parte podrá corresponder a diferentes secciones o capítulos del documento.

Conclusiones. Incluye un breve resumen de lo que se ha presentado en el documento, discutiendo los resultados principales obtenidos y estableciendo líneas de trabajo futuras.

4.2. Prácticas de programación paralela

A continuación se indican qué elementos, además de las recomendaciones generales explicadas antes deben incluirse en el informe de las prácticas de programación paralela.

- Indicar cómo compilar el código.
- Indicar cómo probarlo, tanto para pruebas sencillas como para el análisis
- Indicar las unidades de medida. Por ejemplo, si el tiempo se ha medido en segundos, indicar en la leyenda de la tabla que esa es la unidad, así como en la celda correspondiente. Pe.j., Tiempo (s)
- Si se va a describir la implementación de tu algoritmo, es buena idea apoyarse en diagramas que muestren con un ejemplo sencillo su evolución, relacionándolo con la implementación.
- Utilizar las métricas que se han discutido en este documento para evaluar el rendimiendo de la implementación.
- Mostrar datos con tablas y gráficas.
- Si se han realizado cuestiones o mejores adicionales a lo pedido en el enunciado, ¡es importante comentarlo!

Con respecto a la implementación, para que pueda evaluarse de manera adecuada es importante:

- La salida del programa debe ser clara.
- Añadir nuevos casos de prueba que muestren cómo se comporta tu código. Documentar en la memoria estos casos de prueba y justificarlos.
- Usa herramienta para comprobar que tu programa es correcto. Por ejemplo, para C usa valgrind.
- Comentar el código indicando los aspectos más relevantes de lo que hace función, y en particular ¿por qué? y ¿qué estrategia se sigue?

5. Bibliografía comentada

La sección sobre medidas del rendimiento está basada en en el Capítulo 7 (Performance Analysis) del libro "Parallel Programming in C with MPI and OpenMP" [3].

El libro "How to write technical reports" [2] incluye recomendaciones sobre cómo planificar, estructurar, formatear y crear informes técnicas.

Referencias

[1] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485, 1967.

[2] H. Hering, H. Hering, and Baumann. *How to write technical reports*. Springer, 2019.

[3] M. J. Quinn. Parallel programming in c with mpi and openmp,(2003).