retornamos a los desafios

- selección del algoritmo
- división del trabajo
- distribución de los datos
- sincronización necesaria
- comunicación de los datos entre participantes
- comunicación de los resultados
- medición de características

¿cómo los hemos afrontado?

selección del algoritmo

- hemos usado un algoritmo que en principio nos permite un *speedup* lineal en el número de procesos
  (aunque en realidad mediendo no lo hemos visto...)
- el algoritmo es más rápido para nuestro problema que el algoritmo de la biblioteca estándar
- hemos visto que el algoritmo de la biblioteca tiene un buen *speedup* (si nuestro problema fuese la ordenación general)

Hay que seleccionar el algoritmo adecuado para la situación. Si se usa una biblioteca (trabajos de otros), hay que mirar si usan las características concurrentes del sistema, sino se pierde mucha eficiencia.

división del trabajo

- hemos dividido, con la ayuda del compilador con OpenMP, todos los bucles largos

  ```
  #pragma omp for schedule(static)
  ```

  en trozos de igual tamaño para todos los procesos participantes
- algún código hemos ejecutado con un único proceso

  ```
  #pragma omp single
  ```

distribución de los datos

- estamos en arquitecturas con memoria común
- hemos declarado por defecto ninguna variable compartida y los que nos interesan como compartidas
- constantes están implicitamente compartidas (en OpenMP)
- variables declaradas en el bloque paralelo son automáticamente local al proceso (en OpenMP)
- hemos diseñado el algoritmo de tal manera que los procesos escriben en memoria exclusiva, es decir, nunca escriben en el mismo sitio a la vez

sincronización necesaria

- OpenMP facilita bastante la sincronización simple
- al comienzo del bloque paralelo se crean los procesos cuyo número se ha fijado previamente en algún momento
- al comienzo de un bucle `for` se lanza todos los procesos con la división del trabajo establecida
- al final de un bucle `for` se sincroniza automáticamente con una espera a que todos los participantes hayan terminado
  (nota: eso es cambiable!)
- algunas partes del algoritmo hemos ejecutado a propósito con un único proceso
- al final del bloque paralelo se terminan automáticamente todos los procesos

comunicación de los datos

- no sabemos todavía como el compilador (con OpenMP) consigue que todos los procesos saben que trozos tienen que calcular, por ejemplo, hallar los valores locales de `k` para cada proceso,
- esta comunicación implícita es una de las grandes ventajas de usar OpenMP
- no sabemos todavía como funciona la operación de *reducción* donde todos los procesos participantes consiguen el resultado total de sus valores locales en una variable ompartida
- esta técnica implícita es una de las grandes ventajas de usar OpenMP
- hemos usado una variable booleana compartida `sorted` con modificación posiblemente concurrente
- en este caso no se manifiestan problemas de escrituras concurrentes porque si los procesos escriben algo, entonces escriben lo mismo.

comunicación de los resultados

- el resultado de la ordenación se almacena en una variable compartida
- los paramétros de entrada gestiona un solo proceso y se comunica mediante variables compartidas con los demás
- el resultado del programa (podemos considerar, por ejemplo, la medición del tiempo), escribe un único proceso

los CPUs que hemos usado: i7

Essentials

Export spe

Product Collection	7th Generation Intel® Core™ i7 Processors
Code Name	Products formerly Kaby Lake
Vertical Segment	Mobile
Processor Number	i7-7600U
Status	Launched
Launch Date	Q1'17
Lithography	14 nm
Recommended Customer Price	$393.00
Use Conditions	Industrial Commercial Temp

Performance

# of Cores	2
# of Threads	4
Processor Base Frequency	2.80 GHz
Max Turbo Frequency	3.90 GHz
Cache	4 MB SmartCache
Bus Speed	4 GT/s OPI

los CPUs que hemos usado: i9

Essentials

Export specificatic

Product Collection	Intel® Core™ X-series Processors
Code Name	Products formerly Skylake
Vertical Segment	Desktop
Processor Number	i9-7900X
Status	Launched
Launch Date	Q2'17
Lithography	14 nm
Included Items	Please note: The boxed product does not include a fan or heat sink
Recommended Customer Price	$989.00 - $999.00

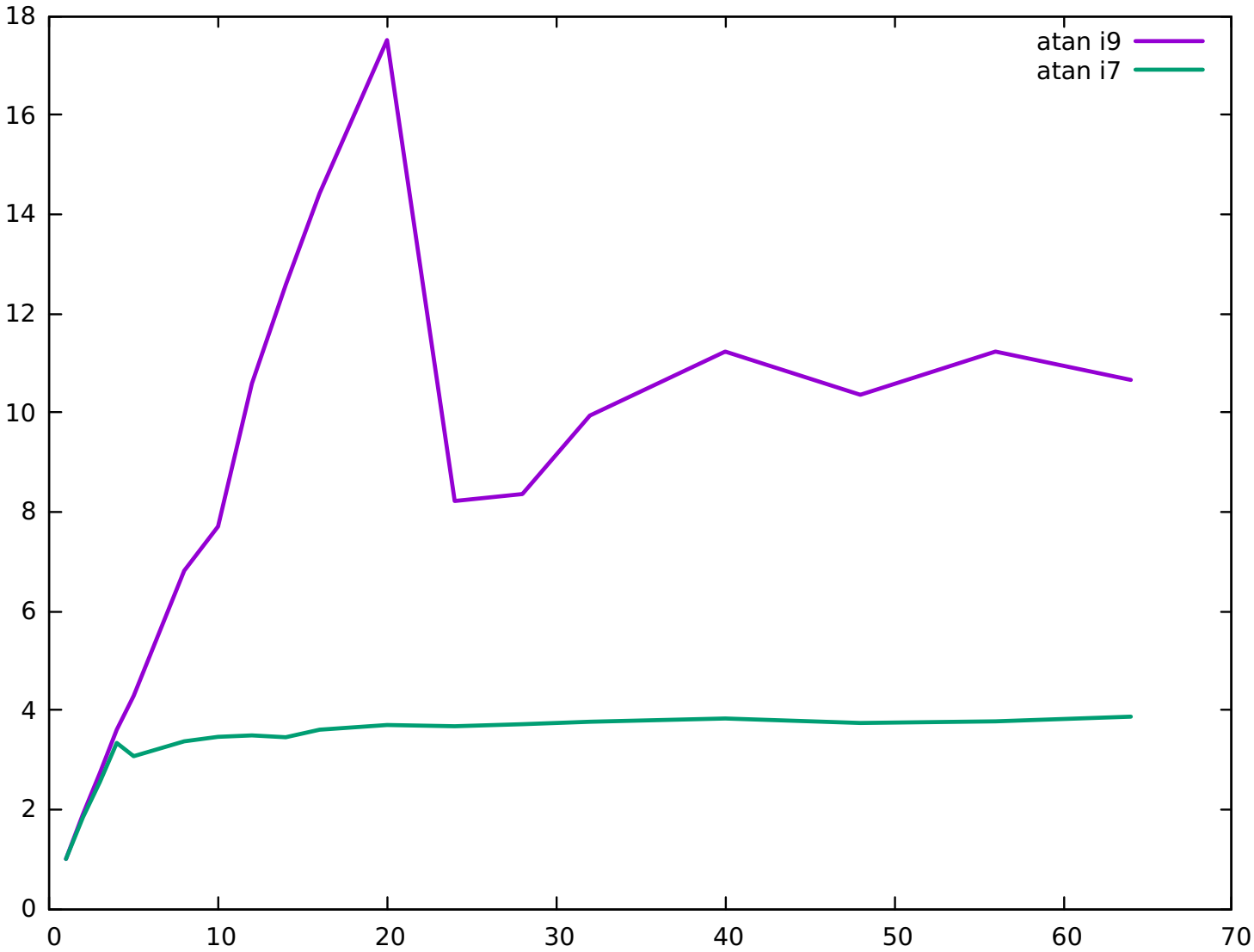
Performance

# of Cores	10
# of Threads	20
Processor Base Frequency	3.30 GHz
Max Turbo Frequency	4.30 GHz
Cache	13.75 MB L3
Bus Speed	8 GT/s DMI3

endendemos las mediciones

- el i7 tiene 2 núcleos pero 4 hilos (físicos)
- el i9 tiene 10 núcleos pero 20 hilos (físicos)
- el *speedup* visto en el algoritmo de la biblioteca refleja el número de núcleos presentes
- el *speedup* entre sistemas (i7 vs. i9) refleja el doble de velocidad del bus
- la ventaja de los hilos físicos no observamos con esta prueba, ya que nuestro problema específico es limitado por acceso a la memoria!

speedup observado en ambos sistemas (prueba de remojo)



interpretación de las mediciones

- en este programa `atan` domina el cálculo
- ambos sistemas alzanzan su mejor rendimiento cuando se trabaja con el número de hilos físicamente disponibles
- en el sistema del i9 baja el rendimiento si superamos con el número de hilos lógicos el número de hilos físicos (seguramente el acoplamiento de los 10 núcleos (20 hilos físicos) a la memoria/cachés no es tan escalable comparado con el i7)
- el *speedup* entre sistemas comparando en el punto de mejor rendimiento llega a un factor de 5.27, es decir, más o menos un factor debido al aumento de hilos físicos y al aumento del reloj de la CPU

reglas del desconfiado...

- Si alguien os vende *speedups* (paralelo a secuencial) muy grandes: quizá su algoritmo secuencial no es el más adecuado para comparar.
- Si veis un *speedup* muy pequeño, pero el algoritmo sugiere más, hay que mirar en el cuadro de comunicación (hacia memoria/almacenamiento o interproceso).
- Hay que tener en cuenta la jerarquia de la memoria en el sistema para decidir la división del trabajo y la distribución de los datos.
- Hay que reducir la necesidad de sincronización a un mínimo posible.
- Hay que intentar solapar cómputo con comunicación de forma equilibrada.

medición de características

- hemos medido solamente el tiempo global de la parte algorítmica del programa desde antes de la creación de los procesos (`pragma omp parallel`) hasta después del bloque en paralelo.
- no sabemos nada en cuanto al tiempo que se necesita para la creación, conmutación, y sincronización de los procesos (observa que a veces hemos tenido más procesos que procesadores (núcleos))
- miramos como podemos medir el tiempo transcurrido individualmente para cada proceso (`Thread`) en Java...