INFORME: TALLER 3

PROGRAMACIÓN FUNCIONAL Y CONCURRENTE

JUAN DIEGO ESCOBAR TRIVIÑO - 2359393

GABRIEL URAZA GARCIA - 2359594

INGENIERIA DE SISTEMAS – 3743

CARLOS ANDRES DELGADO SAAVEDRA

DOCENTE

UNIVERSIDAD DEL VALLE

9/12/2024

TULUA VALLE DEL CAUCA

ÍNDICE GEENERAL

1. INFORME DE PROCESOS
2. INFORME DE CORRECIÓN
3. CONCLUSIONES
4. INFORME DE PROCESOS:

Multiplicación standard secuencial ( def multMatriz)



**taller.App.main(App.scala)** -> Este es el punto de entrada del programa. La función main inicializa la ejecución de la aplicación.

**Java.base/java.lang.Thread.getStackTrace(Thread.java:1619)** -> Genera la pila de llamadas debido al getstackTrace

**Taller.MatrixParallel.multMatriz(MatrixParallel.scala:42)** -> La función multMatriz fue invocada desde otro contexto. En este caso, ejecuta la lógica de multiplicación estándar de matrices.

**Taller.MatrixParallel.StrassenParallel(MatrixParallel.scala:241) ->** La función StrassenParallel llamó a multMatriz. Esto ocurre porque probablemente está utilizando la versión estándar como parte del algoritmo de Strassen.

**taller.App$.$anonfun$benchmarking$6(App.scala:68) ->** Dentro de la función benchmarking, se ejecuta un bloque de código que incluye la llamada a StrassenParallel. Es parte de las pruebas de rendimiento.

**taller.Benchmark.$anonfun$compararAlgoritmo$2(Benchmark.scala:22) ->** función compararAlgoritmo compara la ejecución de dos algoritmos (en este caso, posiblemente la versión estándar y la paralela).

**org.scalameter.MeasureBuilder.$anonfun$measured$1(MeasureBuilder.scala:66)** -> org.scalameter mide el tiempo de ejecución de los algoritmos en este punto.

**org.scalameter.MeasureBuilder.$anonfun$measuredWith$4(MeasureBuilder.scala:54)**

-> Otro nivel de medición y benchmarking, posiblemente asegurando la precisión al incluir múltiples ejecuciones.

**org.scalameter.Measurer$Default.measure(Measurer.scala:133)** -> Realiza una medición concreta del tiempo de ejecución del algoritmo.

**org.scalameter.MeasureBuilder.measuredWith(MeasureBuilder.scala:54) ->** Configura las mediciones de los algoritmos.

**taller.Benchmark.compararAlgoritmo(Benchmark.scala:22)** **-**> Esta función dirige las comparaciones entre algoritmos como parte de las pruebas de rendimiento. **taller.App$.benchmarking(App.scala:64) ->** Coordina la ejecución de las pruebas de rendimiento, incluyendo múltiples tamaños de entrada.

**taller.App$.main(App.scala:44)** -> Regresa al flujo principal de la aplicación luego de las pruebas y cálculos.

Multiplicación de matrices paralela: (def mulMatrizPar)

Texto

Descripción generada automáticamente**java.base/java.lang.Thread.getStackTrace(Thread.java:1619)** ->Se genera la pila de llamadas debido a la ejecución de getStackTrace, mostrando el estado del programa en este momento.

**taller.MatrixParallel.mulMatrizPar(MatrixParallel.scala:69)** ->Se está ejecutando la función mulMatrizPar, que implementa la multiplicación paralela de matrices. Probablemente, fue llamada desde el flujo principal del programa.

**taller.App$.main(App.scala:25)** ->La función main dentro de la clase App llama a mulMatrizPar como parte del flujo principal de la aplicación.

**taller.App.main(App.scala)** ->Este es el punto de entrada del programa. Se ejecuta la función main para iniciar toda la lógica del programa.

Suma De Matrices ( def sumMatriz)Imagen que contiene texto, placa, celular, teléfono

Descripción generada automáticamente

**taller.App.main(App.scala)** -> El punto de entrada del programa llama a la función principal main.

**java.base/java.lang.Thread.getStackTrace(Thread.java:1619)** **->** Se captura la pila de llamadas en este punto con getStackTrace.

**taller.MatrixParallel.SumMatriz(MatrixParallel.scala:74)** **-**> Se ejecuta la función SumMatriz, que suma matrices. Es invocada como parte de la lógica de MultMatrizRec.

**taller.MatrixParallel.MultMatrizRec(MatrixParallel.scala:107)** ->La función recursiva MultMatrizRec se llama a sí misma para resolver la multiplicación de matrices. Esta es una de las llamadas recursivas.

**taller.MatrixParallel.MultMatrizRec(MatrixParallel.scala:107)** ->Otra llamada recursiva de MultMatrizRec, indicando que el problema se sigue descomponiendo en subproblemas más pequeños.

**taller.MatrixParallel.MultMatrizRec(MatrixParallel.scala:105)** ->Aquí comienza otra rama de la recursión, posiblemente alcanzando un caso base o resolviendo subproblemas de menor tamaño.

**taller.App$.$anonfun$benchmarking$2(App.scala:58)** -> Dentro de la función benchmarking, se ejecuta una operación que involucra MultMatrizRec como parte del análisis de rendimiento.

**taller.Benchmark.$anonfun$compararAlgoritmo$1(Benchmark.scala:16)** **-**> La función compararAlgoritmo se encarga de comparar el rendimiento de MultMatrizRec y otros algoritmos de multiplicación.

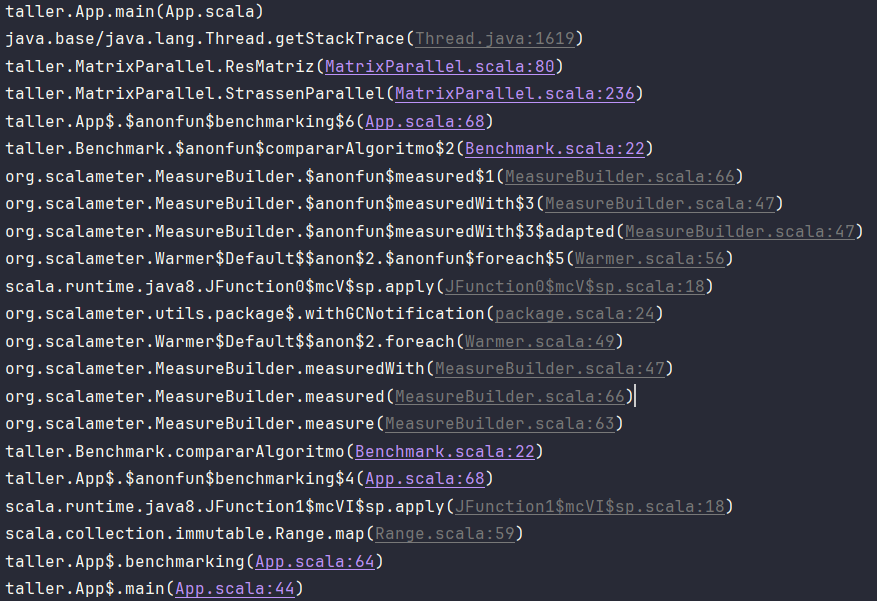
**org.scalameter.MeasureBuilder.$anonfun$measured$1(MeasureBuilder.scala:66)** -> El tiempo de ejecución se mide como parte de las pruebas de rendimiento.

**org.scalameter.MeasureBuilder.$anonfun$measuredWith$3(MeasureBuilder.scala:47 -**> Otra capa de medición, configurando cómo se registran las métricas.

**taller.App$.benchmarking(App.scala:54)** -> Coordina las pruebas y llama a compararAlgoritmo para evaluar la eficiencia de los algoritmos implementados.

**taller.App$.main(App.scala:44)** **-**> Regresa al flujo principal del programa después de completar las pruebas.

Resta de Matrices ( def ResMatriz)



**taller.App.main(App.scala)** -> El programa inicia desde el punto de entrada en la función main.

**java.base/java.lang.Thread.getStackTrace(Thread.java:1619)** -> Se genera y captura la pila de llamadas al ejecutar getStackTrace.

**taller.MatrixParallel.ResMatriz(MatrixParallel.scala:80)** ->La función ResMatriz, que implementa la resta de matrices, está siendo ejecutada. Probablemente es una parte del algoritmo de Strassen.

**taller.MatrixParallel.StrassenParallel(MatrixParallel.scala:236)** ->El algoritmo paralelo de Strassen utiliza ResMatriz para calcular subproblemas como parte de su lógica de descomposición.

**taller.App$.$anonfun$benchmarking$6(App.scala:68)** ->Dentro de la función benchmarking, se invoca StrassenParallel como parte de las pruebas de rendimiento.

**taller.Benchmark.$anonfun$compararAlgoritmo$2(Benchmark.scala:22)** -> La función compararAlgoritmo compara el rendimiento del algoritmo de Strassen con otro algoritmo, ejecutando ambos y registrando los tiempos.

**org.scalameter.MeasureBuilder.$anonfun$measured$1(MeasureBuilder.scala:66)** -> org.scalameter mide el tiempo de ejecución del algoritmo en este punto.

**org.scalameter.MeasureBuilder.$anonfun$measuredWith$3(MeasureBuilder.scala:47)** -> Configura cómo se registran las mediciones para las pruebas de rendimiento.

**taller.App$.benchmarking(App.scala:64)** -> Esta función organiza las pruebas de rendimiento y gestiona la ejecución de los algoritmos a comparar.

**taller.App$.main(App.scala:44)** -> Regresa al flujo principal del programa tras completar las pruebas.

Multiplicación de matrices recursivas ( def MultMatrizRec)

1.4

1. ¿Cual es la implementación mas rápida?

La strassen paralela, por la forma en la que se efectúan las operaciones, al dividirse en operaciones mas senciallas hace que el programa haga las operaciones mas rápidas

1. ¿De que depende que la aceleración sea mejor?

Depende de la forma en la que se tratan los datos al entrar, y el proceso que se hace para las operaciones

1. ¿Puede caracterizar los casos en que es mejor usar la versión secuencial/paralela de cada algoritmo de multiplicación de matrices?

Entre mas grande sea la entrada, mas vale la pena paralelizar, después de matrices de 8x8, vale totalmente la pena paralelizar por el tiempo que se gana, en cambio, en casos anteriores es mucho mejor la forma secuencial.

1.5

¿Será práctico construir versiones de los algoritmos de multiplicación de matrices del enunciado, para la colección ParVector y usar prodPuntoParD en lugar de prodPunto? ¿Por qué si o por qué no?

Solo si los datos son excesivamente grandes valdría la pena, de lo contrario no porque se consumen mas recursos paralelizando y tarda mas tiempo que hacerlo de forma secuencial en casos no tan excesivamente grandes.