Fatine Maataoui **paco1203**
Cinta Gonzalez

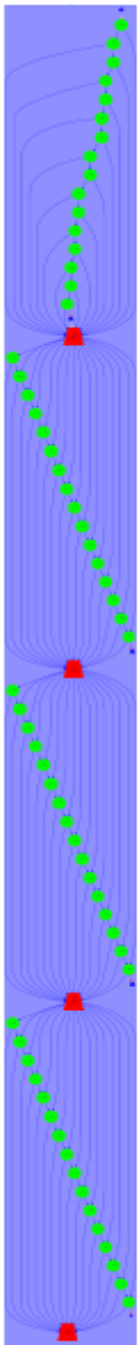
Deliverable

Analysis of task granularities and dependences

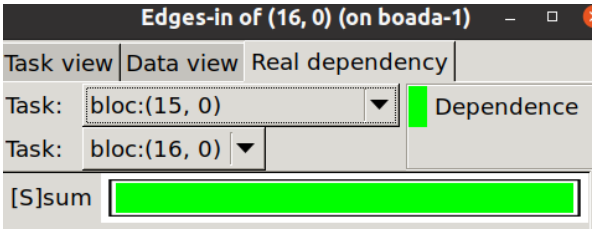
Starting from the initial coarse-grain task decomposition, explain where the calls to the Tareador API have been placed for the fine-grain task decomposition applied to each one of the two solvers: Jacobi and Gauss-Seidel. Explain the original task graph as well as the new graphs that you obtained, reasoning about the causes of the data dependences that appear and how will you protect them in your parallel OpenMP code. Include the timelines with the simulated execution for 4 processors to support your explanations.

Jacobi

1.

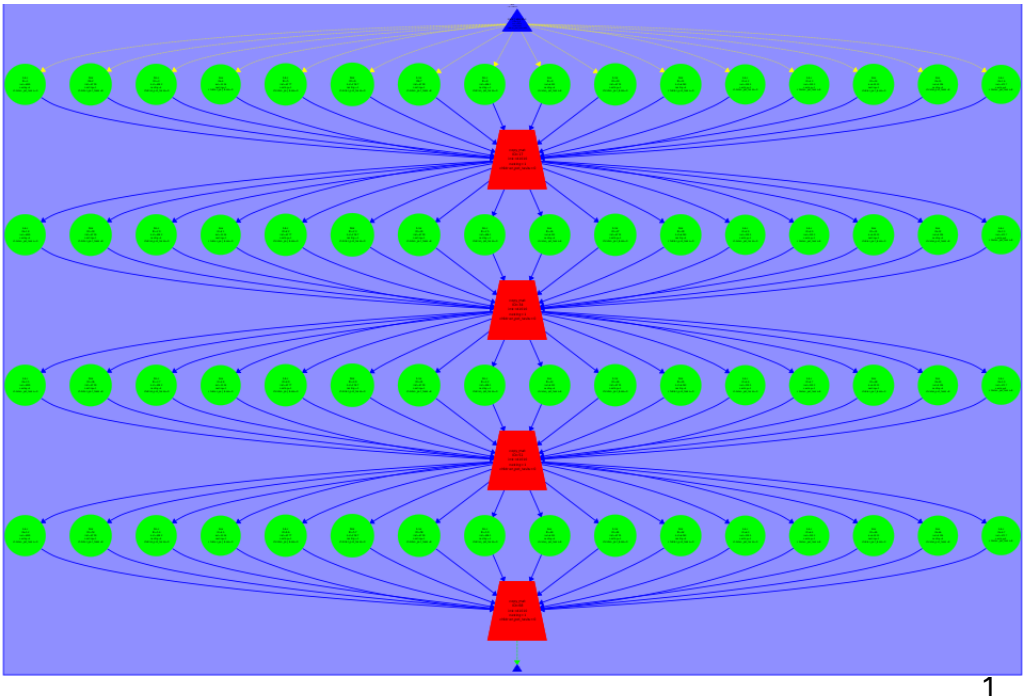


1. Se puede observar como en Jacobi hay una dependencia con todas las iteraciones con la variable **sum.**



Como hay muchas dependencias con la variable **sum**, vamos a arreglarlo con los tareadores que nos ofrecen en la práctica.

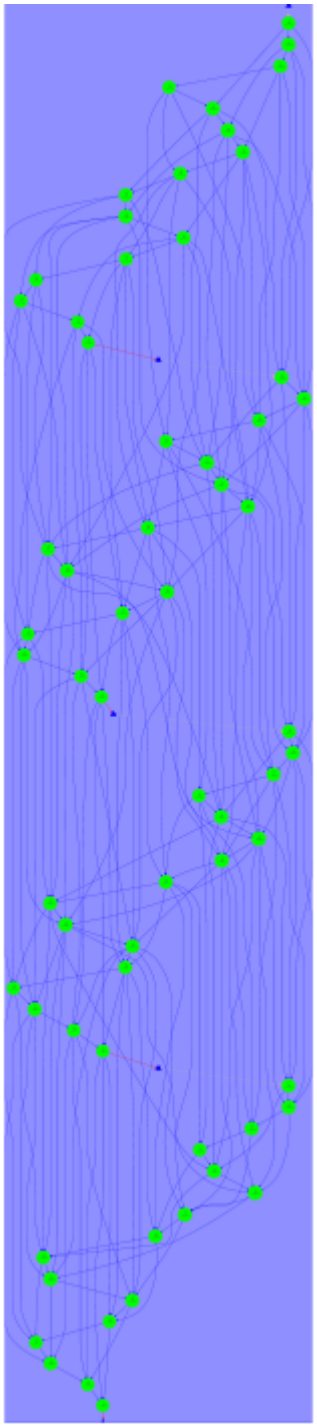
2. En este gráfico se muestra una mejora en la paralelización del programa: ya no tenemos dependencias de la variable suma aunque todavía se puede observar que hay dependencias relacionadas con el orden de acceso a la matriz.



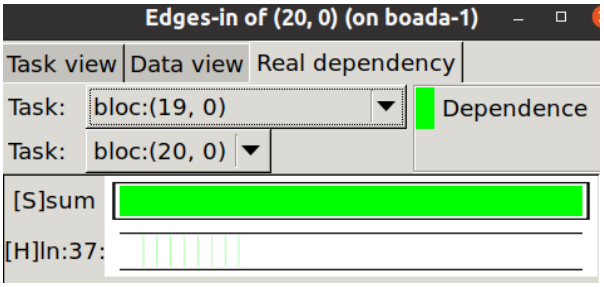
Fatine Maataoui **paco1203**
Cinta Gonzalez

Gauss

1.

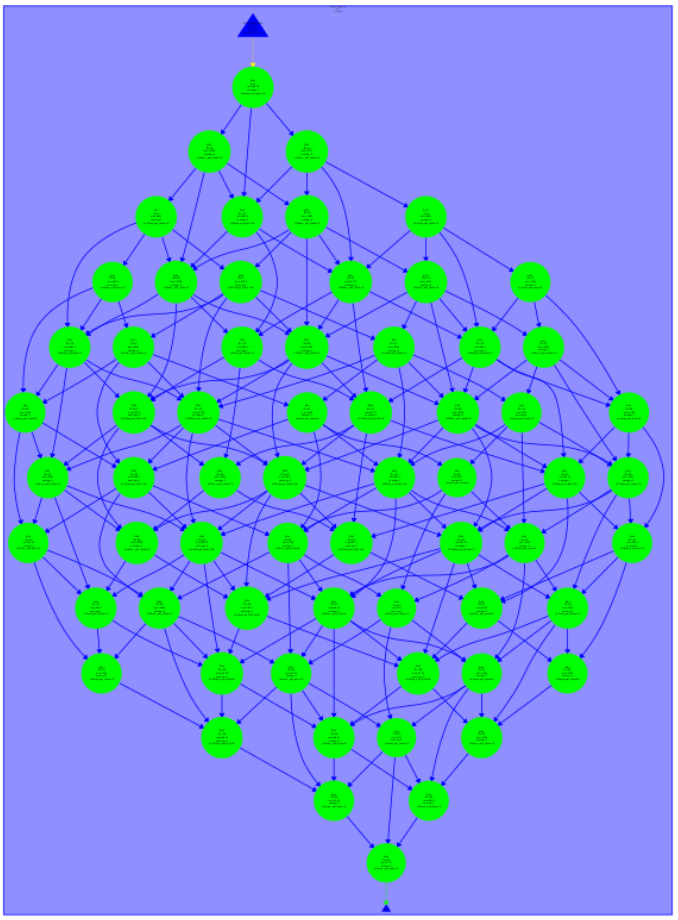


1.Como se puede ver con Gauss hay dependencias también con la variable **sum** y con iteraciones de la matriz, el cual boada la detecta con muchas variables con nombres *extraños*.



Como hay muchas dependencias con la variable **sum**, vamos a arreglarlo con los tareadores que nos ofrecen en la práctica.

2. Al igual que en Jacobi, en este gráfico también se muestra una mejora en la paralelización del programa: no hay dependencias de la variable suma pero si hay dependencias con el orden de acceso a la matriz.



Fatine Maataoui **paco1203**
Cinta Gonzalez

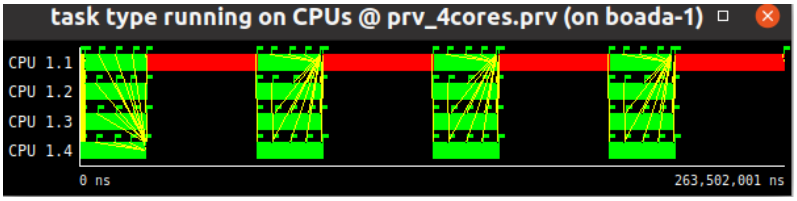
Paraver

Jacobi

Tasks on CPUs profile @ prv_4cores.prv (on boac

Default

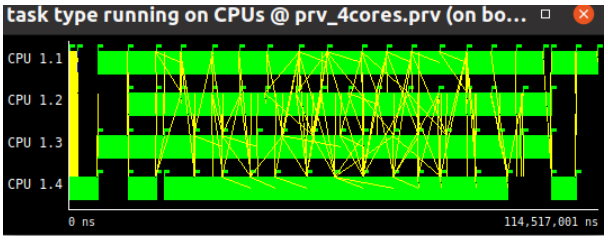
	MAIN_TAREADOR	bloc	copy_mat
CPU 1.1	1	16	4
CPU 1.2	-	16	-
CPU 1.3	-	16	-
CPU 1.4	-	16	-
Total	1	64	4
Average	1	16	4
Maximum	1	16	4
Minimum	1	16	4
StDev	0	0	0
Avg/Max	1	1	1



Gauss

Tasks on CPUs profile @ prv_4cores.prv

	MAIN_TAREADOR	bloc
CPU 1.1	1	18
CPU 1.2	-	15
CPU 1.3	-	16
CPU 1.4	-	15
Total	1	64
Average	1	16
Maximum	1	18
Minimum	1	15
StDev	0	1.22
Avg/Max	1	0.89



Se ve que se crean 4 copias de la matriz.
Con Jacobi se crean el mismo número de tasks por thread y con gauss es más variable.
También con Gauss hay muchas más dependencias ya que tiene una con la matriz porque se accede muchas veces, por tanto, la función **copy_mat** también se podría paralelizar.

Fatine Maataoui **paco1203**
Cinta Gonzalez

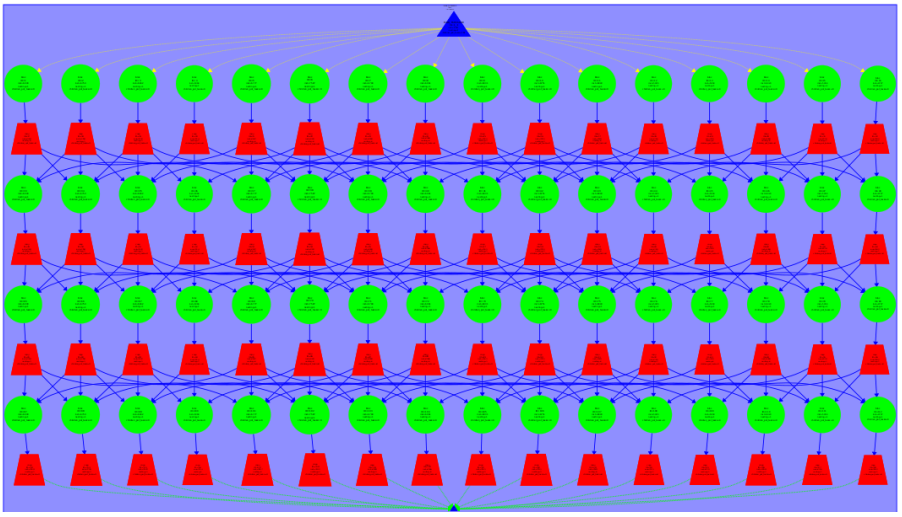
Copy_mat

La función copy_mat **se puede también paralelizar** de la siguiente manera (añadiendo lo que está en rojo para evitar data races):

```
void copy_mat (double *u, double *v, unsigned sizex, unsigned sizey) {

    int nblocksi=4;
    int nblocksj=4;

    for (int blocki=0; blocki<nblocksi; ++blocki) {
      int i_start = lowerb(blocki, nblocksi, sizex);
      int i_end = upperb(blocki, nblocksi, sizex);
      for (int blockj=0; blockj<nblocksj; ++blockj) {

      tareador_start_task("copy");

         int j_start = lowerb(blockj, nblocksj, sizey);
         int j_end = upperb(blockj, nblocksj, sizey);
         for (int i=max(1, i_start); i<=min(sizex-2, i_end); i++)
           for (int j=max(1, j_start); j<=min(sizey-2, j_end); j++)
             v[i*sizey+j] = u[i*sizey+j];

      tareador_end_task("copy");
      }
    }
}
```

Con el grafo de dependencia resultante. Para el caso de **Gauss** es exactamente igual que el **punto 2** del **apartado** de **Gauss**.

Jacobi



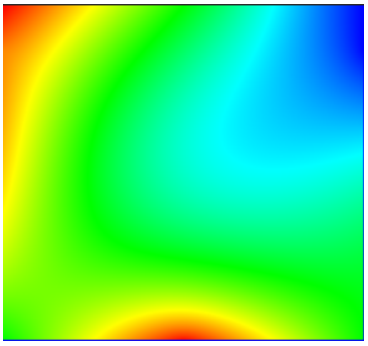
OpenMP parallelization and execution analysis: Jacobi

Describe how did you implement in OpenMP the proposed data decomposition strategy for the heat equation when applying the Jacobi solver, commenting how did you address any detected performance bottleneck (serialisation, load balancing, ...). You should include captures of Paraver windows to justify

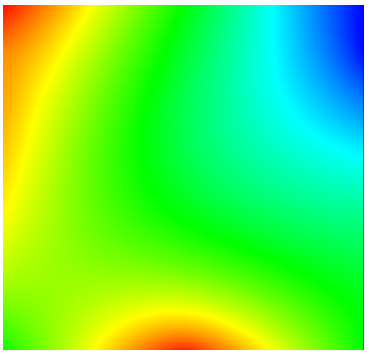
Fatine Maataoui **paco1203**
Cinta Gonzalez

Imágenes de los *Heatmaps* con la implementación secuencial de Jacobi y Gauss:

heat-jacobi.ppm:



heat-gauss.ppm:

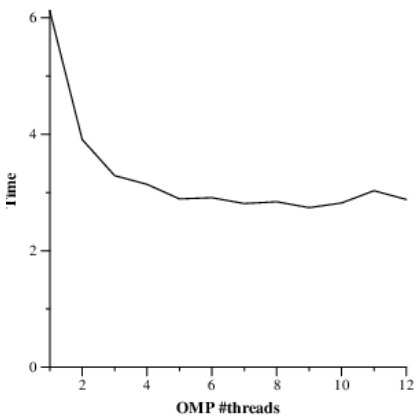


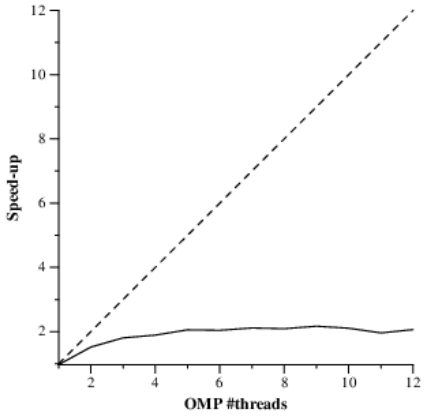
Para readaptar los códigos y que tengan una paralelización más óptima, se han añadido unas condiciones al pragma de la función *solve* y un nuevo pragma a la función *copy_mat*(esta en rojo):

```
void copy_mat (double *u, double *v, unsigned sizex, unsigned sizey) {

    int nblocksi=omp_get_max_threads();
    int nblocksj=1;

  #pragma omp parallel
  {
       int blocki = omp_get_thread_num();
     int i_start = lowerb(blocki, nblocksi, sizex);
     int i_end = upperb(blocki, nblocksi, sizex);
     for (int blockj=0; blockj<nblocksj; ++blockj) {
       int j_start = lowerb(blockj, nblocksj, sizey);
       int j_end = upperb(blockj, nblocksj, sizey);
       for (int i=max(1, i_start); i<=min(sizex-2, i_end); i++)
         for (int j=max(1, j_start); j<=min(sizey-2, j_end); j++)
           v[i*sizey+j] = u[i*sizey+j];
     }
   }
}

double solve (double *u, double *unew, unsigned sizex, unsigned sizey) {
    double tmp, diff, sum=0.0;

    int nblocksi=omp_get_max_threads();
    int nblocksj=1;

  #pragma omp parallel private(diff) reduction(+:sum) // complete data
sharing constructs here
    {
      int blocki = omp_get_thread_num();
      int i_start = lowerb(blocki, nblocksi, sizex);
      int i_end = upperb(blocki, nblocksi, sizex);
      for (int blockj=0; blockj<nblocksj; ++blockj) {
        int j_start = lowerb(blockj, nblocksj, sizey);
        int j_end = upperb(blockj, nblocksj, sizey);
        for (int i=max(1, i_start); i<=min(sizex-2, i_end); i++) {
          for (int j=max(1, j_start); j<=min(sizey-2, j_end); j++) {
                tmp = 0.25 * ( u[ i*sizey          + (j-1) ] +  // left
                           u[ i*sizey      + (j+1) ] +  // right
                           u[ (i-1)*sizey + j      ] +  // top
                           u[ (i+1)*sizey + j      ] ); // bottom
                diff = tmp - u[i*sizey+ j];
                sum += diff * diff;
                unew[i*sizey+j] = tmp;
          }
        }
      }
    }

    return sum;
```

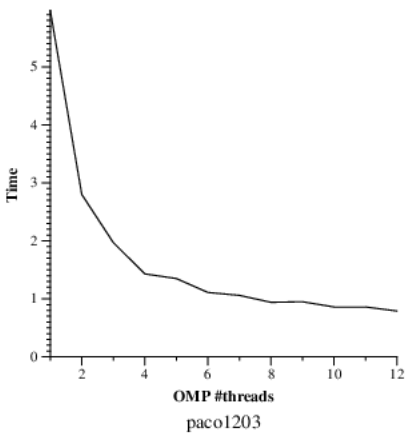
Tal y como se aprecia en las 4 gráficas, se aprecia como en las gráficas del segundo caso (*Gráficas una vez **paralelizado el código***) el speed-up del **código secuencial** se acerca mucho a cero y en el **paralizado** se aproxima más a lo “ideal”, por lo tanto se observa que hay una mejora bastante grande a la hora de paralizar.

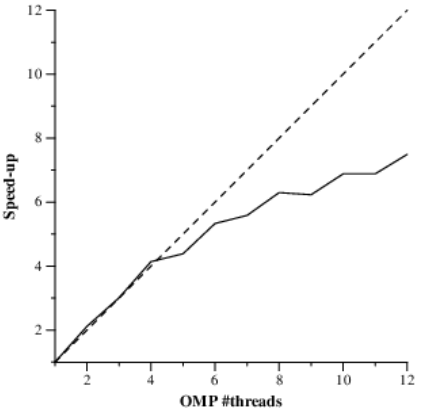
Gráficas speed-up antes de modificar la paralelización:





Gráficas una vez paralelizado el código:





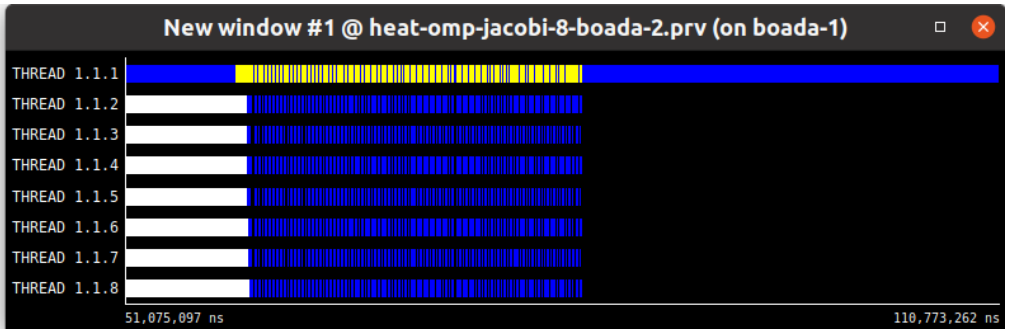
Fatine Maataoui **paco1203**
Cinta Gonzalez

Paraver

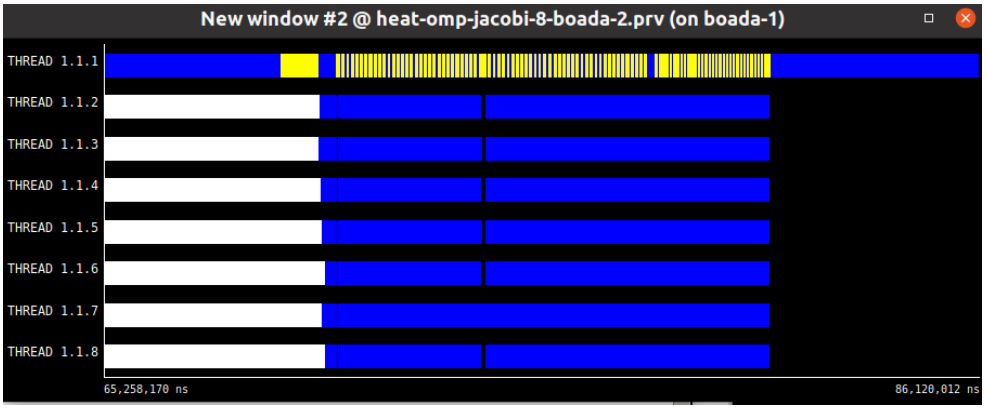
En ésta sección se puede observar, en color azul, como se está ejecutando cada procesador y cómo procede la paralelización.

Con el código **paralizado** se contempla como la ejecuciónes en paralelo están mucho más repartidas, es decir, las barras de color azul son prácticamente iguales con la barra amarilla. Por lo que hace a la eficiencia acaba de confirmar lo dicho en la frase anterior, todos los threads, excepto uno, trabajan casi por igual, es decir, todos tienen el mismo porcentaje (se puede observar en la tabla del estado del thread).

Con el código original:



Con el código paralizadoo:



Con el código original:

2D thread state profile @ heat-omp-jacobi-8-boada-2.prv (on boada-1)

Default

	Running	Not created	Scheduling and Fork/Join	I/O	Others
THREAD 1.1.1	98.60 %	-	1.38 %	0.02 %	0.00 %
THREAD 1.1.2	10.59 %	89.36 %	-	0.05 %	-
THREAD 1.1.3	7.53 %	92.43 %	-	0.04 %	-
THREAD 1.1.4	10.48 %	89.48 %	-	0.04 %	-
THREAD 1.1.5	7.48 %	92.48 %	-	0.04 %	-
THREAD 1.1.6	10.53 %	89.43 %	-	0.04 %	-
THREAD 1.1.7	7.67 %	92.29 %	-	0.04 %	-
THREAD 1.1.8	10.08 %	89.88 %	-	0.04 %	-
Total	162.96 %	635.35 %	1.38 %	0.31 %	0.00 %
Average	20.37 %	90.76 %	1.38 %	0.04 %	0.00 %
Maximum	98.60 %	92.48 %	1.38 %	0.05 %	0.00 %
Minimum	7.48 %	89.36 %	1.38 %	0.02 %	0.00 %
StDev	29.60 %	1.42 %	0 %	0.01 %	0 %
Avg/Max	0.21	0.98	1	0.79	1

Con el código modificado:

2D thread state profile @ heat-omp-jacobi-8-boada-2.prv (on boada-1)

Default

	Running	Not created	Scheduling and Fork/Join	I/O	Others
THREAD 1.1.1	99.07 %	-	0.90 %	0.02 %	0.00 %
THREAD 1.1.2	9.34 %	90.60 %	-	0.05 %	-
THREAD 1.1.3	9.25 %	90.70 %	-	0.04 %	-
THREAD 1.1.4	9.31 %	90.64 %	-	0.05 %	-
THREAD 1.1.5	9.17 %	90.79 %	-	0.04 %	-
THREAD 1.1.6	9.13 %	90.82 %	-	0.05 %	-
THREAD 1.1.7	9.26 %	90.70 %	-	0.04 %	-
THREAD 1.1.8	9.08 %	90.87 %	-	0.05 %	-
Total	163.62 %	635.13 %	0.90 %	0.35 %	0.00 %
Average	20.45 %	90.73 %	0.90 %	0.04 %	0.00 %
Maximum	99.07 %	90.87 %	0.90 %	0.05 %	0.00 %
Minimum	9.08 %	90.60 %	0.90 %	0.02 %	0.00 %
StDev	29.72 %	0.09 %	0 %	0.01 %	0 %
Avg/Max	0.21	1.00	1	0.83	1