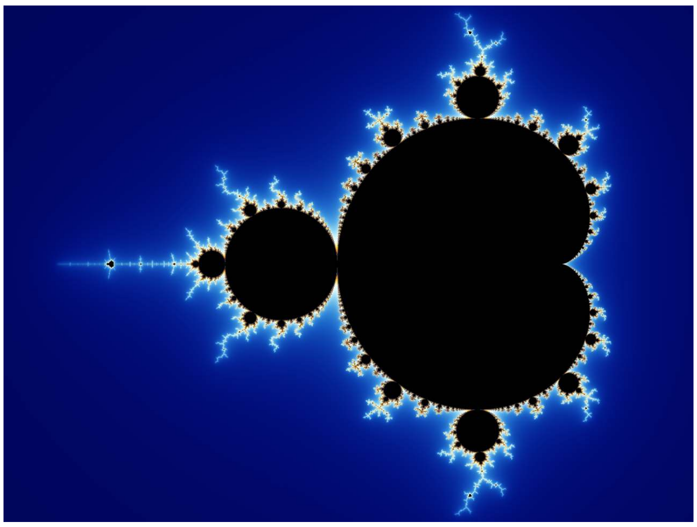
PARreportlab3.pdf

msi
Pack portátil + monitor portátil
Más posibilidades, más productividad, más portátil. Más de todo.
Si una sola pantalla te sabe a poco, MSI tiene el dúo dinámico que necesitas. Un combo portátil + monitor extra para que tu setup de trabajo te siga a todos lados como tu ex stalkeándote. Más espacio, más productividad, menos excusas.

Universitat Politècnica de Catalunya

PARALLELISM

Laboratory practice nº3: The Mandelbrot set



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Index

1 Iterative task decomposition analysis

1. 1 Original version:

Code: No modifications, it's the original one.

Tareador TDG:

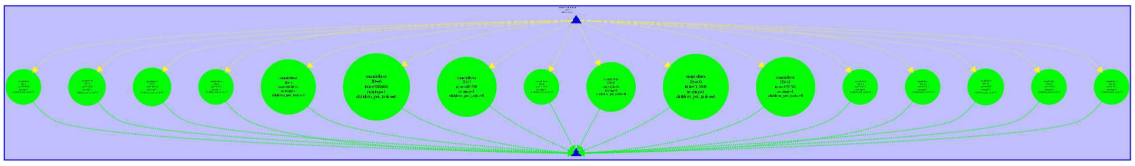


Figure 1: Image of the original's version TDG

Dependence Analysis: There aren't dependencies between the tasks mandelbrot executes.

Tareador TDG (without dependences): there's no possible elimination of dependencies.

T ∞ Analysis: If we add 16 threads, as there are 16 tasks that can be executed in parallel, then we will obtain the T∞, which will have the value of the task that executes the most points in the region with maximum number of iterations.

1. 2 Original version with -d:

Code: No modifications, is the original.

Tareador TDG:

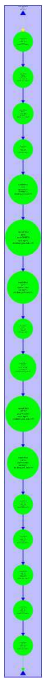


Figure 2: Image of the original's version with -d flag activated TDG

Dependence Analysis: there are dependencies due to the execution of this two lines of code that call to Xwindows:

- XSetForeground (display, gc, color);
- XDrawPoint (display, win, gc, px, py);

The first line of code, sets the default color to paint the pixels of the background while the second line, calls a function which reads the color set by default and decides which will be the final color for the pixel (px,py), that's why there's a dependency between them, due to the variable color, named as X11_COLOR_fake.

The dependency could be removed by protecting the variable color, using a critical, in our case we've used:

- tareador_disable_object(&X11_COLOR_fake);
- tareador_enable_object(&X11_COLOR_fake);

to remove the dependency, to obtain the TDG without dependencies shown below.

Tareador TDG (without dependences):

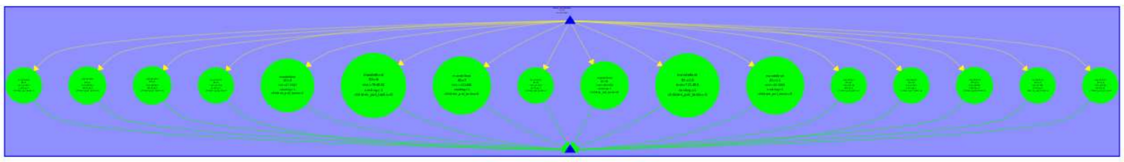


Figure 3: Image of the original's version with -d flag activated TDG, without dependeces

T ∞ Analysis: If we add 16 threads, as there are 16 tasks that can be executed in parallel, then we will obtain the T∞, which will have the value of the task that executes the most points in the region with maximum number of iterations. (Idem original version without flags)

1. 3 Original version with -h:

Code: No modifications, it's the original one.

Tareador TDG:

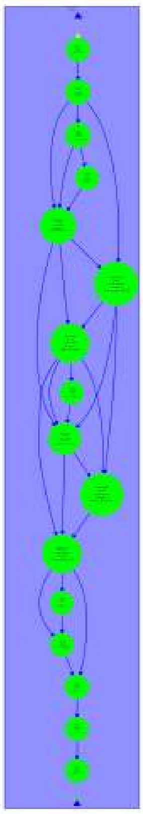


Figure 4: Image of the original's version with -h flag activated TDG

msi®

Más posibilidades,
más productividad,
más portátil.
Más de todo.

Pack portátil +
monitor portátil

Si una sola pantalla te sabe a poco, MSI tiene el dúo dinámico que necesitas. Un combo portátil + monitor extra para que tu setup de trabajo te siga a todos lados como tu ex stalkeándote. Más espacio, más productividad, menos excusas.

Dependence Analysis: We can see that there are dependencies due to the execution of this line of code:

- histogram[M[py][px]-1]++;

This line of code, specifically the variable histogram, is the one that produces dependencies between tasks since there's an access to a position of the variable histogram that can be modified before, therefore a dependency with the task that has modified it before is needed to respect the data race.

The dependency can be removed by protecting the variable histogram itself, using a critical, in our case we've used:

- tareador_disable_object(histogram);
- tareador_enable_object(histogram);

to remove the dependency, to obtain the TDG without dependencies shown below.

Tareador TDG (without dependences):

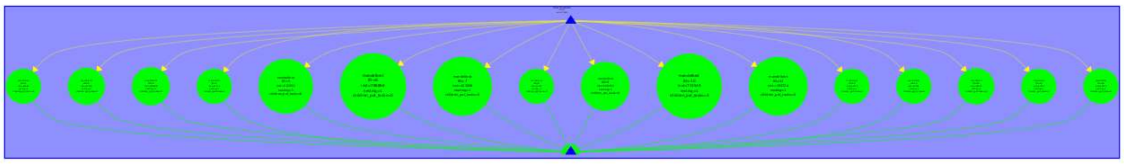
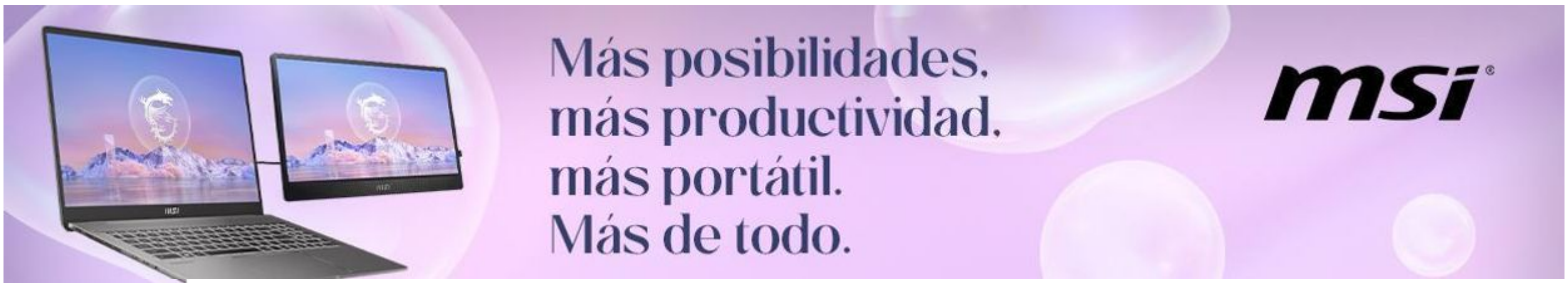


Figure 5: Image of the original's version with -h flag activated TDG, without dependeces

T ∞ Analysis: If we add 16 threads, as there are 16 tasks that can be executed in parallel, then we will obtain the T∞, which will have the value of the task that executes the most points in the region with maximum number of iterations. (Idem original version without flags)



1. 4 Finer grain version:

<u>Code</u>:

C/C++

```
void mandel_tiled(int M[ROWS][COLS], double CminR, double CminI, double CmaxR,
                               double CmaxI, double scale_real, double scale_imag, int maxiter)
{
    int equal1, equal2;

    for (int y = 0; y < ROWS; y += TILE)
    {
                for (int x = 0; x < COLS; x += TILE)
                {
                               equal1 = 1;
                               equal2 = 1;
                               // check horizontal borders
                               tareador_start_task("horizontal");
                               for (int px = x; px < x + TILE; px++)
                               {
                               M[y][px] = pixel_dwell(COLS, ROWS, CminR, CminI, CmaxR, CmaxI, px, y, scale_real, scale_imag, maxiter);
                               M[y + TILE - 1][px] = pixel_dwell(COLS, ROWS, CminR, CminI, CmaxR, CmaxI, px, y + TILE - 1, scale_real, scale_imag, maxiter);
                               equal1 = equal1 && (M[y][x] == M[y][px]);
                               equal1 = equal1 && (M[y][x] == M[y + TILE - 1][px]);
                               }
                               tareador_end_task("horizontal");
                               // check vertical borders
                               tareador_start_task("vertical");
                               for (int py = y; py < y + TILE; py++)
                               {
                               M[py][x] = pixel_dwell(COLS, ROWS, CminR, CminI, CmaxR, CmaxI, x, py, scale_real, scale_imag, maxiter);
                               M[py][x + TILE - 1] = pixel_dwell(COLS, ROWS, CminR, CminI, CmaxR, CmaxI, x + TILE - 1, py, scale_real, scale_imag, maxiter);
                               equal2 = equal2 && (M[y][x] == M[py][x]);
                               equal2 = equal2 && (M[y][x] == M[py][x + TILE - 1]);
                               }
                               tareador_end_task("vertical");
                               // check if we can avoid computation of tile
                               tareador_start_task("computation");
                               if (equal1 && equal2 && M[y][x] == maxiter)
                               {
                               // fill all with the same value
                               for (int py = y; py < y + TILE; py++) {
                                                             tareador_start_task("if");
                               for (int px = x; px < x + TILE; px++)
                               {
                               M[py][px] = M[y][x];
                               if (output2histogram)
                                               histogram[M[py][px] - 1]++;
                               if (output2display)
                               {
                                               /* Scale color and display point  */
                                               long color = (long)((M[py][px] - 1) * scale_color) + min_color;
                                               if (setup_return == EXIT_SUCCESS)
                                               {
                                               XSetForeground(display, gc, color);
                                               XDrawPoint(display, win, gc, px, py);
                                               }
                               }
                               }
                                               tareador_end_task("if");
                                               }
                               }
                               else
                               {
                               // computation of tile
                               for (int py = y; py < y + TILE; py++) {
                                                             tareador_start_task("else");
                               for (int px = x; px < x + TILE; px++)
                               {
                               M[py][px] = pixel_dwell(COLS, ROWS, CminR, CminI, CmaxR, CmaxI, px, py, scale_real, scale_imag, maxiter);
                               if (output2histogram)
                                               histogram[M[py][px] - 1]++;
                               if (output2display)
                               {
                                               /* Scale color and display point  */
                                               long color = (long)((M[py][px] - 1) * scale_color) + min_color;
                                               if (setup_return == EXIT_SUCCESS)
                                               {
                                               XSetForeground(display, gc, color);
                                               XDrawPoint(display, win, gc, px, py);
                                               }
                               }
                               }
                                               tareador_end_task("else");
                                               }
                               }
                               tareador_end_task("computation");
                }
    }
}
```

<u>Tareador TDG</u>:

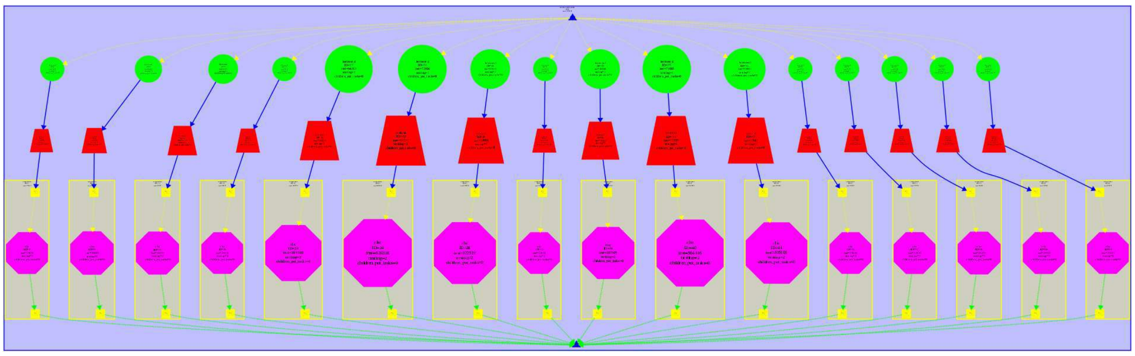


Figure 6: Image of the finer grain's version TDG, with dependeces

Dependence Analysis: (In order to ease the readability we've not included all tasks in the TDG, in all the following explanations and analysis we've included these tasks)
We can see that the dependent version is sequential as all tasks depend on the main at first (blue triangle), and then followed up in all cases by horizontal borders check task (green) and finally vertical borders check task (red) before arriving at the computation (yellow rectangle). This is because of the variable "equal" that was used in both horizontal and vertical checks.

Tareador TDG (without dependences): To remove the dependencies we need to see that the variable responsible of those dependencies is the variable equal, which is used by the vertical and the horizontal tasks, therefore we just need to create two equals (equals1 and equals2) in order to use them on in each tasks (vertical and horizontal) and finally in the part of computation we must check that both equals are positive to go inside the if sentence.

We can see that the "else" tasks after removing the dependences with the strategy of having two equals, the "else" tasks just depend on the horizontal tasks in all the executions except the last execution which also depend on the vertical task.

Here we can see the TDG:

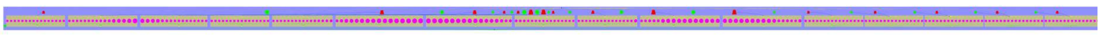


Figure 7: Image of the TDG without dependences

T ∞ Analysis: If we add 40 threads, we see that the execution of the code exploits at its maximum the parallelism, since if we execute the Tareador with 128 threads we obtain the same time of execution (see bottom right).

At least we see that we need 32 threads due to that we have 16 vertical tasks and 16 horizontal tasks, but if we put 32 threads we see that there are 4 horizontal tasks and 4 vertical tasks that have a longer time of execution and therefore we can't start executing else tasks at the same time than the other 12 tasks of each type, that's why we need 8 more threads to let them execute these highly demanding tasks in terms of time, letting the other threads start executing the else tasks before to exploit the maximum amount of parallelism.

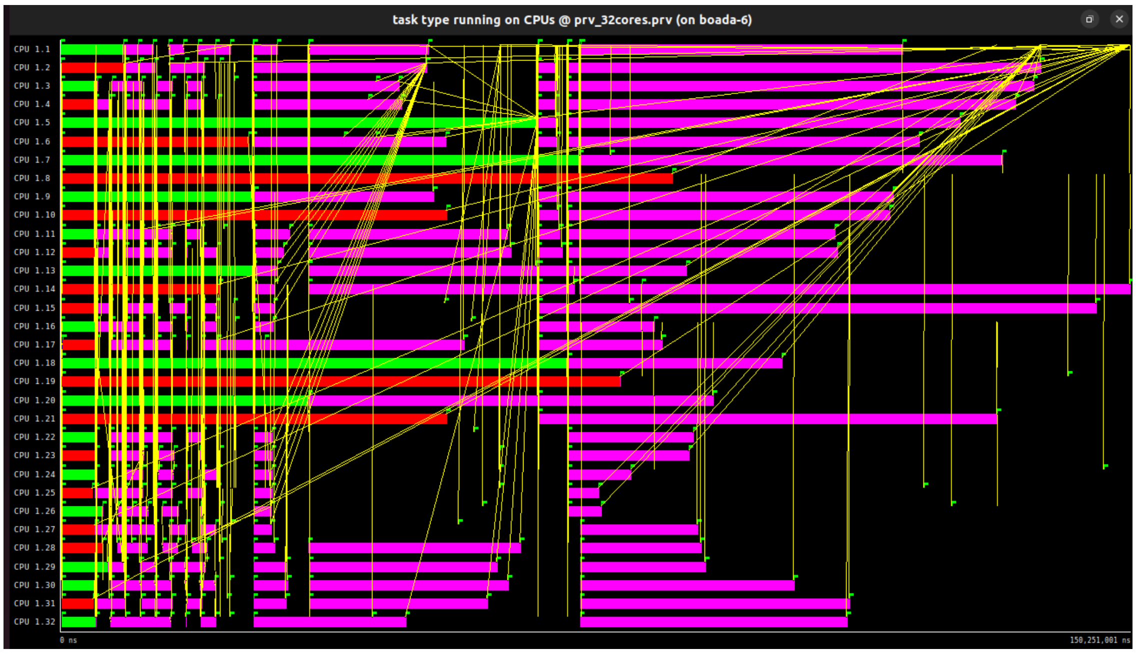


Figure 8: Image of the execution with 32 cores

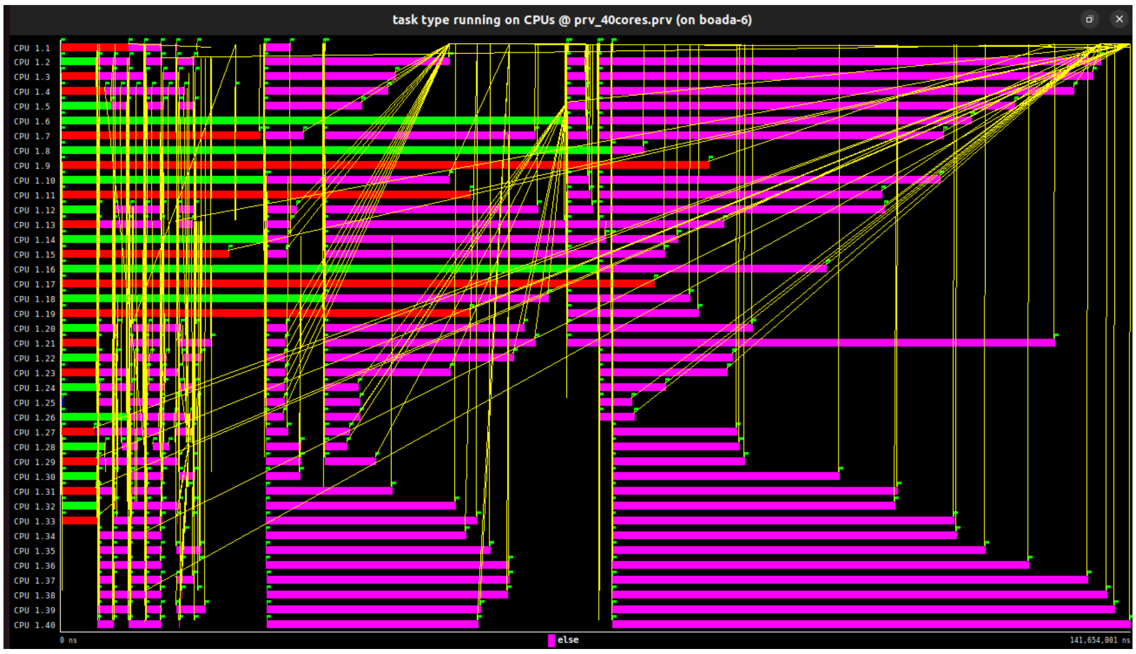


Figure 9: Image of the execution with 40 cores

1. 5 Column of tiles version:

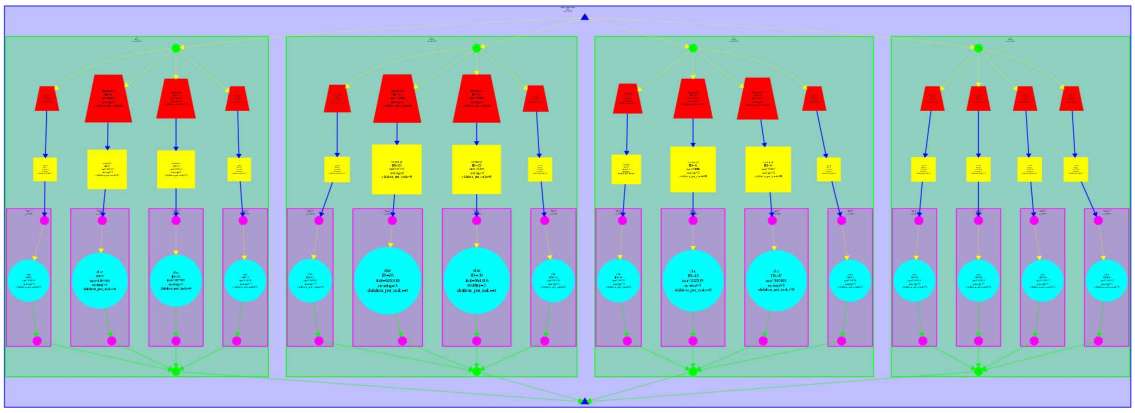
Code:

C/C++

```
void mandel_tiled(int M[ROWS][COLS], double CminR, double CminI, double CmaxR, double CmaxI, double scale_real, double scale_imag, int maxiter)
{
        int equal1, equal2;
        for (int x = 0; x < COLS; x += TILE)
        {
         tareador_start_task("column");
         for (int y = 0; y < ROWS; y += TILE)
         {
                equal1 = 1;
                equal2 = 1;
                // check horizontal borders
                tareador_start_task("horizontal");
                for (int px = x; px < x + TILE; px++)
                {
                        M[y][px] = pixel_dwell(COLS, ROWS, CminR, CminI, CmaxR, CmaxI, px, y, scale_real, scale_imag, maxiter);
                        M[y + TILE - 1][px] = pixel_dwell(COLS, ROWS, CminR, CminI, CmaxR, CmaxI, px, y + TILE - 1, scale_real, scale_imag, maxiter);
                        equal1 = equal1 && (M[y][x] == M[y][px]);
                        equal1 = equal1 && (M[y][x] == M[y + TILE - 1][px]);
                }
                tareador_end_task("horizontal");
                // check vertical borders
                tareador_start_task("vertical");
                for (int py = y; py < y + TILE; py++)
                {
                        M[py][x] = pixel_dwell(COLS, ROWS, CminR, CminI, CmaxR, CmaxI, x, py, scale_real, scale_imag, maxiter);
                        M[py][x + TILE - 1] = pixel_dwell(COLS, ROWS, CminR, CminI, CmaxR, CmaxI, x + TILE - 1, py, scale_real, scale_imag, maxiter);
                        equal2 = equal2 && (M[y][x] == M[py][x]);
                        equal2 = equal2 && (M[y][x] == M[py][x + TILE - 1]);
                }
                tareador_end_task("vertical");
                // check if we can avoid computation of tile
                tareador_start_task("computation");
                if (equal1 && equal2 && M[y][x] == maxiter)
                {
                        // fill all with the same value
                        for (int py = y; py < y + TILE; py++) {
                                tareador_start_task("if");
                                for (int px = x; px < x + TILE; px++)
                                {
                                        M[py][px] = M[y][x];
                                        if (output2histogram)
                                                histogram[M[py][px] - 1]++;
                                        if (output2display)
                                        {
                                                /* Scale color and display point  */
                                                long color = (long)((M[py][px] - 1) * scale_color) + min_color;
                                                if (setup_return == EXIT_SUCCESS)
                                                {
                                                        XSetForeground(display, gc, color);
                                                        XDrawPoint(display, win, gc, px, py);
                                                }
                                        }
                                }
                                tareador_end_task("if");
                        }
                }
                else
                {
                        // computation of tile
                        for (int py = y; py < y + TILE; py++) {
                                tareador_start_task("else");
                                for (int px = x; px < x + TILE; px++)
                                {
                                        M[py][px] = pixel_dwell(COLS, ROWS, CminR, CminI, CmaxR, CmaxI, px, py, scale_real, scale_imag,
maxiter);

                                        if (output2histogram)
                                                histogram[M[py][px] - 1]++;
                                        if (output2display)
                                        {
                                                /* Scale color and display point  */
                                                long color = (long)((M[py][px] - 1) * scale_color) + min_color;
                                                if (setup_return == EXIT_SUCCESS)
                                                {
                                                        XSetForeground(display, gc, color);
                                                        XDrawPoint(display, win, gc, px, py);
                                                }
                                        }
                                }
                                tareador_end_task("else");
                        }
                }
                tareador_end_task("computation");
         }
         tareador_end_task("column");
        }
}
```

Tareador TDG:



Dependence Analysis: (In order to ease the readability we've not included all tasks in the TDG, in all the following explanations and analysis we've included these tasks) (Idem as finer grain with the addition of the task englobing columns)
As we can observe, the dependent version is quite sequential as all tasks depend on the main at first (blue triangle), and then followed up all the time by horizontal borders check task (red) and finally vertical borders check task (yellow) before arriving at the computation (purple rectangle). This is because of the variable "equal" that was used in both horizontal and vertical checks.

Tareador TDG (without dependences): (Idem as strategy for finer grain) To remove the dependencies we need to see that the variable responsible of those dependencies is the variable equal, which is used by the vertical and the horizontal tasks, therefore we just need to create two equals (equals1 and equals2) in order to use them on in each tasks (vertical and horizontal) and finally in the part of computation we must check that both equals are positive to go inside the if sentence.

We can see that the "else" tasks after removing the dependences with the strategy of having two equals, the "else" tasks just depend from the horizontal tasks in all the executions except the last execution which also depend from the vertical task.

Here we can see the TDG:

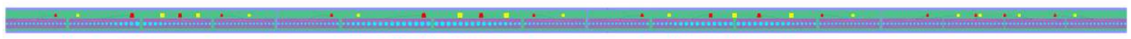


Figure 11: Image of the TDG without dependences

T ∞ Analysis: (Idem as finer grain analysis) If we add 40 threads, we see that the execution of the code exploits at its maximum the parallelism, since if we execute the Tareador with 128 threads we obtain the same time of execution (see bottom right).

At least we see that we need 32 threads due to that we have 16 vertical tasks and 16 horizontal tasks, but if we put 32 threads we see that there are 4 horizontal tasks and 4 vertical tasks that have a longer time of execution and therefore we can't start executing else tasks at the same time than the other 12 tasks of each type, that's why we need 8 more threads to let them execute these highly demanding tasks in terms of time, letting the other threads start executing the else tasks before to exploit the maximum amount of parallelism.

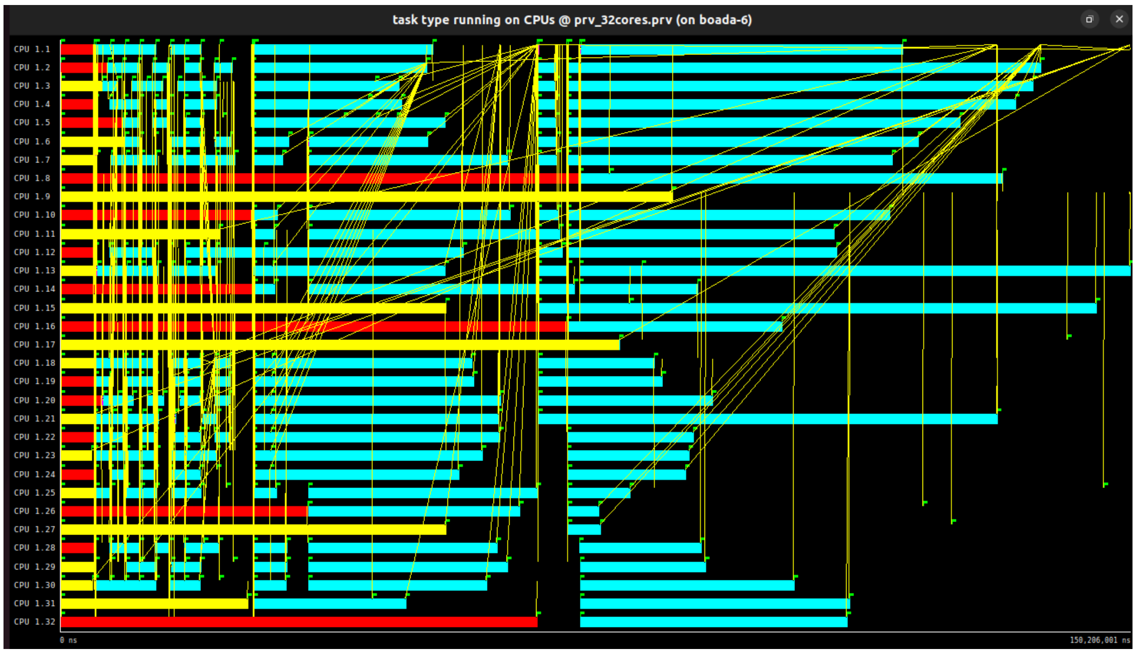


Figure 12: Image of the execution with 32 cores

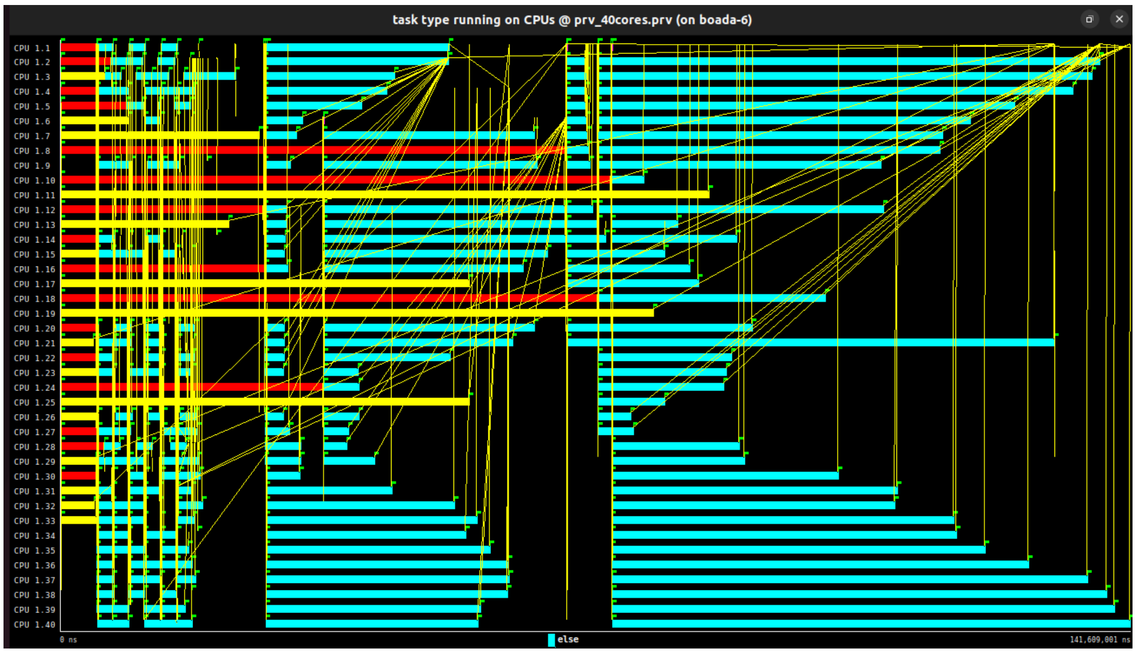
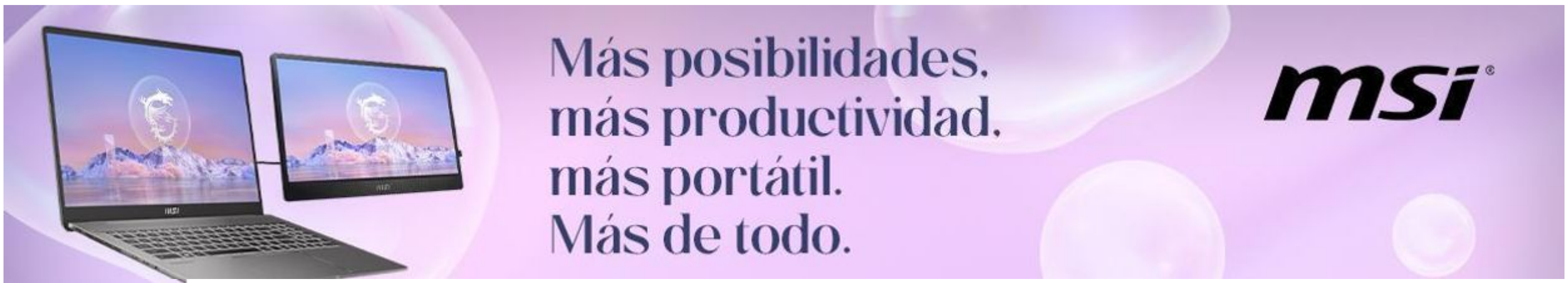


Figure 13: Image of the execution with 40 cores



2 Recursive task decomposition analysis

2. 1 Recursive Leaf Strategy version

<u>Code</u>: We've added the Tareador tasks for the leaf at the base case to implement the leaf strategy

C/C++

```c
void mandel_tiled_rec(int M[ROWS][COLS], int NRows, int NCols, int start_fil, int start_col, double CminR, double CminI, double CmaxR, double CmaxI,double scale_real, double
scale_imag, int maxiter)
{
  int equal;
  int x, y;
  equal = 1;
  y = start_fil;
  x = start_col;
  //check horizontal borders
  for (int px=x; px<x+NCols; px++) {
             M[y][px]       = pixel_dwell(COLS, ROWS, CminR, CminI, CmaxR, CmaxI, px, y, scale_real, scale_imag, maxiter);
             M[y+NRows-1][px] = pixel_dwell(COLS, ROWS, CminR, CminI, CmaxR, CmaxI, px, y+NRows-1, scale_real, scale_imag, maxiter);
                     equal = equal && (M[y][x] == M[y][px]);
             equal = equal && (M[y][x] == M[y+NRows-1][px]);
  }
  //check vertical borders
  for (int py=y; py<y+NRows; py++){
             M[py][x]       = pixel_dwell(COLS, ROWS, CminR, CminI, CmaxR, CmaxI, x, py, scale_real, scale_imag, maxiter);
             M[py][x+NCols-1] = pixel_dwell(COLS, ROWS, CminR, CminI, CmaxR, CmaxI, x+NCols-1, py, scale_real, scale_imag, maxiter);
             equal = equal && (M[y][x] == M[py][x]);
             equal = equal && (M[y][x] == M[py][x+NCols-1]);
  }
  //check if we can avoid computation of tile
  if (equal && M[y][x]==maxiter)
  {
             // fill all with the same value: base case
             for (int py=y; py<y+NRows; py++)
             for (int px=x; px<x+NCols; px++)
             {
             M[py][px] = M[y][x];
                if (output2histogram)
                            histogram[M[y][x]-1]++;
                if (output2display)
                            {
                            /* Scale color and display point  */
                            long color = (long) ((M[y][x]-1) * scale_color) + min_color;
                            if (setup_return == EXIT_SUCCESS)
                             {
                               XSetForeground (display, gc, color);
                               XDrawPoint (display, win, gc, px, py);
                             }
                            }
             }
  }
  else {
             if (NCols <= TILE) {
    tareador_start_task("leaf");
      // computation of tile: base case
             for (int py=y; py<y+NRows; py++)
             for (int px=x; px<x+NCols; px++)
             {
             M[py][px] = pixel_dwell(COLS, ROWS, CminR, CminI, CmaxR, CmaxI, px, py, scale_real, scale_imag, maxiter);
                            if (output2histogram)
                            histogram[M[py][px]-1]++;
             if (output2display)
             {
             /* Scale color and display point  */
             long color = (long) ((M[py][px]- 1) * scale_color) + min_color;
             if (setup_return == EXIT_SUCCESS)
                               {
                            XSetForeground (display, gc, color);
                            XDrawPoint (display, win, gc, px, py);
             }
             }
             }
    tareador_end_task("leaf");
             }
             else
             {
      // computation of tile: recursive cases
             if (NRows > TILE)
             {
             mandel_tiled_rec(M, NRows/2, NCols/2, start_fil, start_col, CminR, CminI, CmaxR, CmaxI, scale_real, scale_imag, maxiter);
             mandel_tiled_rec(M, NRows/2, NCols/2, start_fil, start_col+NCols/2, CminR, CminI, CmaxR, CmaxI, scale_real, scale_imag, maxiter);
             mandel_tiled_rec(M, NRows/2, NCols/2, start_fil+NRows/2, start_col, CminR, CminI, CmaxR, CmaxI, scale_real, scale_imag, maxiter);
             mandel_tiled_rec(M, NRows/2, NCols/2, start_fil+NRows/2, start_col+NCols/2, CminR, CminI, CmaxR, CmaxI, scale_real, scale_imag, maxiter);
             }
             else
             {
             mandel_tiled_rec(M, NRows, NCols/2, start_fil, start_col, CminR, CminI, CmaxR, CmaxI, scale_real, scale_imag, maxiter);
             mandel_tiled_rec(M, NRows, NCols/2, start_fil, start_col+NCols/2, CminR, CminI, CmaxR, CmaxI, scale_real, scale_imag, maxiter);
             }

             }
  }
}
```

Tareador TDG:

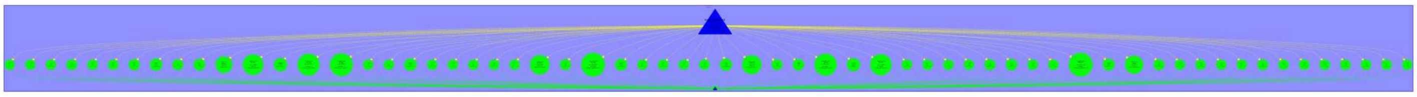


Figure 14: Image leaf strategy's version

Dependence Analysis: As we can observe, there are no dependencies beyond all the tasks, just that all of them need the main.

Tareador TDG (without dependences): There's no possibility to eliminate dependencies as there aren't any.

T ∞ Analysis: We see that the number of threads that are capable to fulfill the parallelism is 3, since if we use just two threads, there will be one thread executing the main task and another executing the leaf tasks, but the one executing the leaf tasks is not capable to exploit the maximum parallelism, therefore another extra thread is needed and the final number of threads is 3.

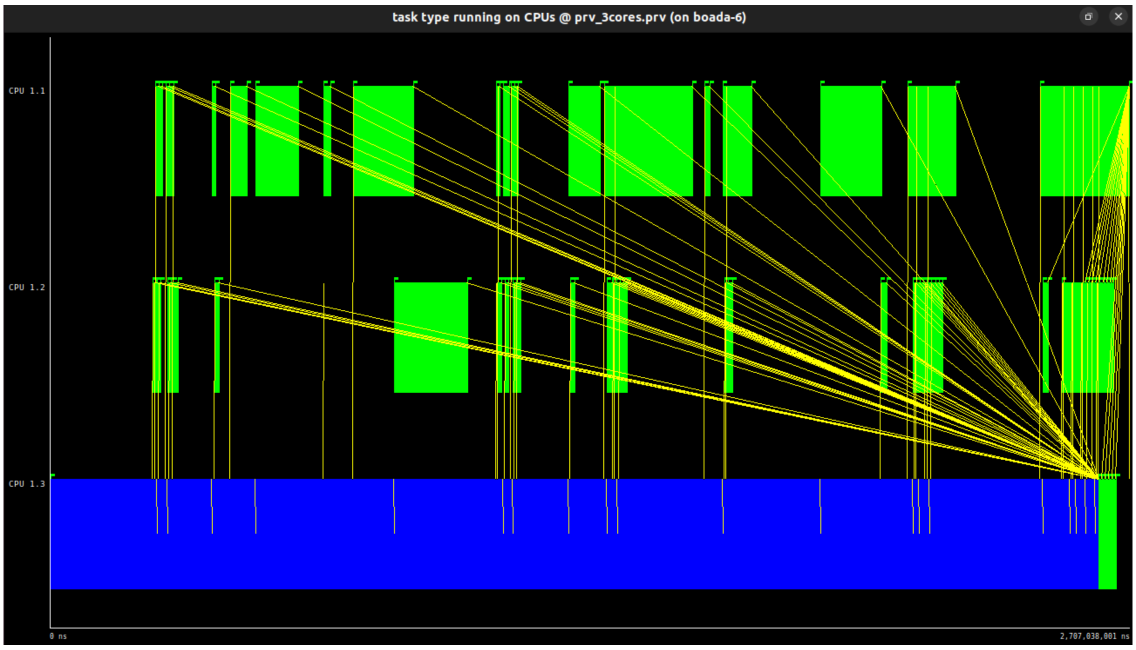


Figure 15: Image of leaf's strategy version execution with 3 cores

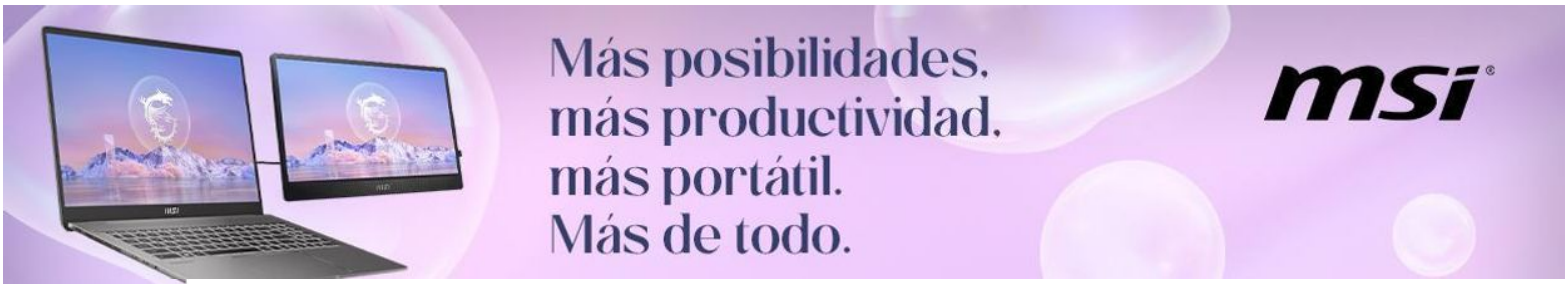
2. 2 Recursive Leaf Strategy version

Code: We've added the Tareador tasks for the leaf at the recursive case to implement the tree strategy.

C/C++

```c
void mandel_tiled_rec(int M[ROWS][COLS], int NRows, int NCols, int start_fil, int start_col, double CminR, double CminI, double CmaxR, double CmaxI,double scale_real, double
scale_imag, int maxiter)
{
  int equal;
  int x, y;
  equal = 1;
  y = start_fil;
  x = start_col;
  //check horizontal borders
  for (int px=x; px<x+NCols; px++) {
             M[y][px]       = pixel_dwell(COLS, ROWS, CminR, CminI, CmaxR, CmaxI, px, y, scale_real, scale_imag, maxiter);
             M[y+NRows-1][px] = pixel_dwell(COLS, ROWS, CminR, CminI, CmaxR, CmaxI, px, y+NRows-1, scale_real, scale_imag, maxiter);
                             equal = equal && (M[y][x] == M[y][px]);
             equal = equal && (M[y][x] == M[y+NRows-1][px]);
  }
  //check vertical borders
  for (int py=y; py<y+NRows; py++){
             M[py][x]       = pixel_dwell(COLS, ROWS, CminR, CminI, CmaxR, CmaxI, x, py, scale_real, scale_imag, maxiter);
             M[py][x+NCols-1] = pixel_dwell(COLS, ROWS, CminR, CminI, CmaxR, CmaxI, x+NCols-1, py, scale_real, scale_imag, maxiter);
             equal = equal && (M[y][x] == M[py][x]);
             equal = equal && (M[y][x] == M[py][x+NCols-1]);
  }
  //check if we can avoid computation of tile
  if (equal && M[y][x]==maxiter)
  {
             // fill all with the same value: base case
             for (int py=y; py<y+NRows; py++)
             for (int px=x; px<x+NCols; px++)
             {
             M[py][px] = M[y][x];
                if (output2histogram)
                            histogram[M[y][x]-1]++;
                if (output2display)
                            {
                            /* Scale color and display point  */
                            long color = (long) ((M[y][x]-1) * scale_color) + min_color;
                            if (setup_return == EXIT_SUCCESS)
                             {
                               XSetForeground (display, gc, color);
                               XDrawPoint (display, win, gc, px, py);
                             }
                            }
             }
  }
  else {
             if (NCols <= TILE)
             {
      // computation of tile: base case
             for (int py=y; py<y+NRows; py++)
             for (int px=x; px<x+NCols; px++)
             {
             M[py][px] = pixel_dwell(COLS, ROWS, CminR, CminI, CmaxR, CmaxI, px, py, scale_real, scale_imag, maxiter);
                            if (output2histogram)
                            histogram[M[py][px]-1]++;
             if (output2display)
             {
             /* Scale color and display point  */
             long color = (long) ((M[py][px]- 1) * scale_color) + min_color;
             if (setup_return == EXIT_SUCCESS)
                               {
                            XSetForeground (display, gc, color);
                            XDrawPoint (display, win, gc, px, py);
             }
             }
             }
             }
             else
             {
      // computation of tile: recursive cases
             if (NRows > TILE)
             {
             tareador_start_task("tree1");
             mandel_tiled_rec(M, NRows/2, NCols/2, start_fil, start_col, CminR, CminI, CmaxR, CmaxI, scale_real, scale_imag, maxiter);
             tareador_end_task("tree1");
             tareador_start_task("tree2");
             mandel_tiled_rec(M, NRows/2, NCols/2, start_fil, start_col+NCols/2, CminR, CminI, CmaxR, CmaxI, scale_real, scale_imag, maxiter);
             tareador_end_task("tree2");
             tareador_start_task("tree3");
             mandel_tiled_rec(M, NRows/2, NCols/2, start_fil+NRows/2, start_col, CminR, CminI, CmaxR, CmaxI, scale_real, scale_imag, maxiter);
             tareador_end_task("tree3");
             tareador_start_task("tree4");
             mandel_tiled_rec(M, NRows/2, NCols/2, start_fil+NRows/2, start_col+NCols/2, CminR, CminI, CmaxR, CmaxI, scale_real, scale_imag, maxiter);
             tareador_end_task("tree4");
             }
             else
             {
             mandel_tiled_rec(M, NRows, NCols/2, start_fil, start_col, CminR, CminI, CmaxR, CmaxI, scale_real, scale_imag, maxiter);
             mandel_tiled_rec(M, NRows, NCols/2, start_fil, start_col+NCols/2, CminR, CminI, CmaxR, CmaxI, scale_real, scale_imag, maxiter);
             }

             }
  }
}
```



Tareador TDG:

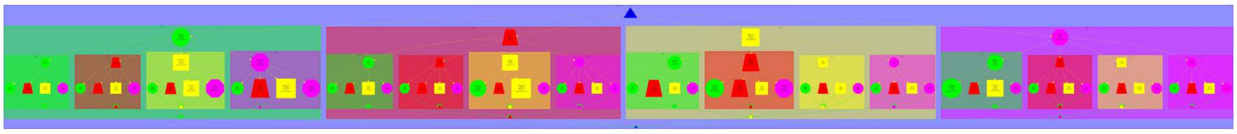


Figure 16: Image of tree's strategy version TDG

Dependence Analysis: As we can observe, there are dependences, as expected in a tree strategy since in this case we create 4 tasks for each execution of the function, which recursively will create 4 more and so forth.

Tareador TDG (without dependences): As in this version we are doing a tree strategy, there's no possibility to eliminate dependencies since the resulting TDG reflects accurately that the dependencies between tasks must be respected among them.

T ∞ Analysis: In order to exploit the maximum parallelism we see that the number of threads needed are 12, since as every task takes a different number of t.u. to execute, the number of processors that exploit the maximum parallelism is 12.

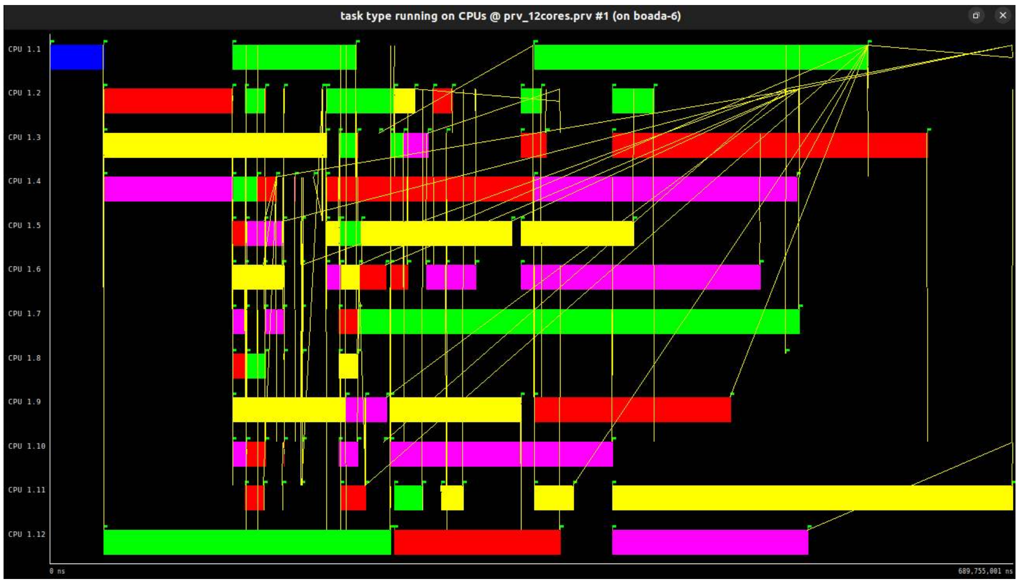


Figure 17: Image of tree's strategy version execution with 12 cores

3. Summary of the different parallelism performances

Version	T_1	T_∞	Parallelism
Iterative: original strategy (no parameters) Iterative: original strategy (-d) Iterative: original strategy (-h)	3.178.018.001 ns 3.305.594.001 ns 3.244.674.001 ns	786.956,001 ns 786.956,001 ns 786.956,001 ns	4,04 4,2 4,12
Iterative: Finer grain	3.178.618.001 ns	141.684.001 ns	22,43
Iterative: Column of tiles	3.178.618.001 ns	141.609.001 ns	22,44
Recursive: Leaf	4.545.464.001 ns	2.797.038.001 ns	1,65
Recursive: Tree	4.545.464.001 ns	689.755.001 ns	6,59