Walkthrough: Debugging a Parallel Application

Visual Studio 2015

The new home for Visual Studio documentation is Visual Studio 2017 Documentation on docs.microsoft.com.

The latest version of this topic can be found at Walkthrough: Debugging a Parallel Application.

This walkthrough shows how to use the **Parallel Tasks** and **Parallel Stacks** windows to debug a parallel application. These windows help you understand and verify the runtime behavior of code that uses the Task Parallel Library (TPL) or the Concurrency Runtime. This walkthrough provides sample code that has built-in breakpoints. After the code breaks, the walkthrough shows how to use the **Parallel Tasks** and **Parallel Stacks** windows to examine it.

This walkthrough teaches these tasks:

- How to view the call stacks of all threads in one view.
- How to view the list of `System.Threading.Tasks.Task` instances that are created in your application.
- How to view the real call stacks of tasks instead of threads.
- How to navigate to code from the **Parallel Tasks** and **Parallel Stacks** windows.
- How the windows cope with scale through grouping, zooming, and other related features.

Prerequisites

This walkthrough assumes that **Just My Code** is enabled. On the **Tools** menu, click **Options**, expand the **Debugging** node, select **General**, and then select **Enable Just My Code (Managed only)**. If you do not set this feature, you can still use this walkthrough, but your results may differ from the illustrations.

C# Sample

If you use the C# sample, this walkthrough also assumes that External Code is hidden. To toggle whether external code is displayed, right-click the **Name** table header of the **Call Stack** window, and then select or clear **Show External Code**. If you do not set this feature, you can still use this walkthrough, but your results may differ from the illustrations.

Illustrations

The illustrations in this topic recorded on a quad core computer running the C# sample. Although you can use other configurations to complete this walkthrough, the illustrations may differ from what is displayed on your computer.

Creating the Sample Project

The sample code in this walkthrough is for an application that does nothing. The goal is just to understand how to use the tool windows to debug a parallel application.

To create the sample project

1. In Visual Studio, on the **File** menu, point to **New** and then click **Project**.
2. In the **Installed Templates** pane, select either Visual C#, Visual Basic, or Visual C++. For the managed languages, ensure that .NET Framework 4 is displayed in the framework box.
3. Select **Console Application** and then click **OK**. Remain in Debug configuration, which is the default.
4. Open the .cpp, .cs, or .vb code file in the project. Delete its contents to create an empty code file.
5. Paste the following code for your chosen language into the empty code file.

C#
```
using System;
using System.Threading;
using System.Threading.Tasks;
using System.Diagnostics;

class S
{
  static void Main()
  {
    pcount = Environment.ProcessorCount;
    Console.WriteLine("Proc count = " + pcount);
    ThreadPool.SetMinThreads(4, -1);
    ThreadPool.SetMaxThreads(4, -1);

    t1 = new Task(A, 1);
    t2 = new Task(A, 2);
    t3 = new Task(A, 3);
    t4 = new Task(A, 4);
    Console.WriteLine("Starting t1 " + t1.Id.ToString());
    t1.Start();
    Console.WriteLine("Starting t2 " + t2.Id.ToString());
    t2.Start();
    Console.WriteLine("Starting t3 " + t3.Id.ToString());
    t3.Start();
    Console.WriteLine("Starting t4 " + t4.Id.ToString());
    t4.Start();

    Console.ReadLine();
  }

  static void A(object o)
  {
    B(o);
  }
  static void B(object o)
  {
    C(o);
  }
  static void C(object o)
  {
```

```
  int temp = (int)o;

  Interlocked.Increment(ref aa);
  while (aa < 4)
  {
    ;
  }

  if (temp == 1)
  {
    // BP1 - all tasks in C
    Debugger.Break();
    waitFor1 = false;
  }
  else
  {
    while (waitFor1)
    {
      ;
    }
  }
  switch (temp)
  {
    case 1:
      D(o);
      break;
    case 2:
      F(o);
      break;
    case 3:
    case 4:
      I(o);
      break;
    default:
      Debug.Assert(false, "fool");
      break;
  }
}
static void D(object o)
{
  E(o);
}
static void E(object o)
{
  // break here at the same time as H and K
  while (bb < 2)
  {
    ;
  }
  //BP2 - 1 in E, 2 in H, 3 in J, 4 in K
  Debugger.Break();
  Interlocked.Increment(ref bb);

  //after
  L(o);
}
static void F(object o)
{
  G(o);
}
static void G(object o)
{
  H(o);
}
static void H(object o)
```

```
{
  // break here at the same time as E and K
  Interlocked.Increment(ref bb);
  Monitor.Enter(mylock);
  while (bb < 3)
  {
    ;
  }
  Monitor.Exit(mylock);


  //after
  L(o);
}
static void I(object o)
{
  J(o);
}
static void J(object o)
{
  int temp2 = (int)o;

  switch (temp2)
  {
    case 3:
      t4.Wait();
      break;
    case 4:
      K(o);
      break;
    default:
      Debug.Assert(false, "fool2");
      break;
  }
}
static void K(object o)
{
  // break here at the same time as E and H
  Interlocked.Increment(ref bb);
  Monitor.Enter(mylock);
  while (bb < 3)
  {
    ;
  }
  Monitor.Exit(mylock);


  //after
  L(o);
}
static void L(object oo)
{
  int temp3 = (int)oo;

  switch (temp3)
  {
    case 1:
      M(oo);
      break;
    case 2:
      N(oo);
      break;
    case 4:
      O(oo);
      break;
```

```
        default:
          Debug.Assert(false, "fool3");
          break;
      }
    }
    static void M(object o)
    {
      // breaks here at the same time as N and Q
      Interlocked.Increment(ref cc);
      while (cc < 3)
      {
        ;
      }
      //BP3 - 1 in M, 2 in N, 3 still in J, 4 in O, 5 in Q
      Debugger.Break();
      Interlocked.Increment(ref cc);
      while (true)
        Thread.Sleep(500); // for ever
    }
    static void N(object o)
    {
      // breaks here at the same time as M and Q
      Interlocked.Increment(ref cc);
      while (cc < 4)
      {
        ;
      }
      R(o);
    }
    static void O(object o)
    {
      Task t5 = Task.Factory.StartNew(P, TaskCreationOptions.AttachedToParent);
      t5.Wait();
      R(o);
    }
    static void P()
    {
      Console.WriteLine("t5 runs " + Task.CurrentId.ToString());
      Q();
    }
    static void Q()
    {
      // breaks here at the same time as N and M
      Interlocked.Increment(ref cc);
      while (cc < 4)
      {
        ;
      }
      // task 5 dies here freeing task 4 (its parent)
      Console.WriteLine("t5 dies " + Task.CurrentId.ToString());
      waitFor5 = false;
    }
    static void R(object o)
    {
      if ((int)o == 2)
      {
        //wait for task5 to die
        while (waitFor5) { ;}


        int i;
        //spin up all procs
        for (i = 0; i < pcount - 4; i++)
        {
          Task t = Task.Factory.StartNew(() => { while (true);});
          Console.WriteLine("Started task " + t.Id.ToString());
        }

        Task.Factory.StartNew(T, i + 1 + 5, TaskCreationOptions.AttachedToParent);
  //scheduled
        Task.Factory.StartNew(T, i + 2 + 5, TaskCreationOptions.AttachedToParent);
  //scheduled
        Task.Factory.StartNew(T, i + 3 + 5, TaskCreationOptions.AttachedToParent);
  //scheduled
        Task.Factory.StartNew(T, i + 4 + 5, TaskCreationOptions.AttachedToParent);
  //scheduled
        Task.Factory.StartNew(T, (i + 5 + 5).ToString(),
  TaskCreationOptions.AttachedToParent); //scheduled

        //BP4 - 1 in M, 2 in R, 3 in J, 4 in R, 5 died
        Debugger.Break();
      }
      else
      {
        Debug.Assert((int)o == 4);
        t3.Wait();
      }
    }
    static void T(object o)
    {
      Console.WriteLine("Scheduled run " + Task.CurrentId.ToString());
    }
    static Task t1, t2, t3, t4;
    static int aa = 0;
    static int bb = 0;
    static int cc = 0;
    static bool waitFor1 = true;
    static bool waitFor5 = true;
    static int pcount;
    static S mylock = new S();
  }
```

1. On the **File** menu, click **Save All**.
2. On the **Build** menu, click **Rebuild Solution**.

   Notice that there are four calls to `Debugger.Break` (`DebugBreak` in the C++ sample) Therefore, you do not have to insert breakpoints; just running the application will cause it to break in the debugger up to four times.

Using the Parallel Stacks Window: Threads View

On the **Debug** menu, click **Start Debugging**. Wait for the first breakpoint to be hit.

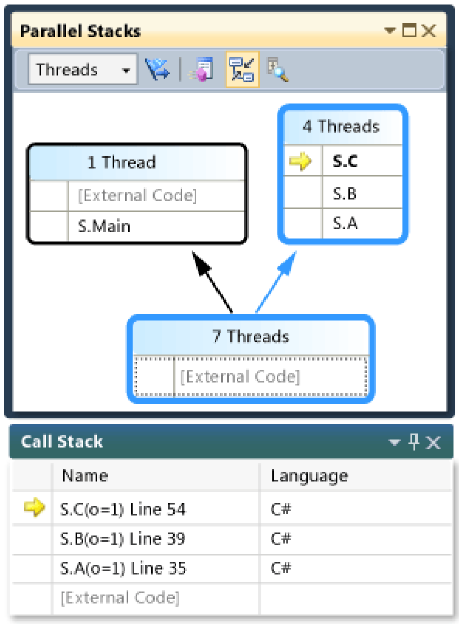
To view the call stack of a single thread

1. On the **Debug** menu, point to **Windows** and then click **Threads**. Dock the **Threads** window at the bottom of Visual Studio.
2. On the **Debug** menu, point to **Windows** and then click **Call Stack**. Dock the **Call Stack** window at the bottom Visual Studio.
3. Double-click a thread in the **Threads** window to make it current. Current threads have a yellow arrow. When you change the current thread, its call stack is displayed in the **Call Stack** window.

To examine the Parallel Stacks window

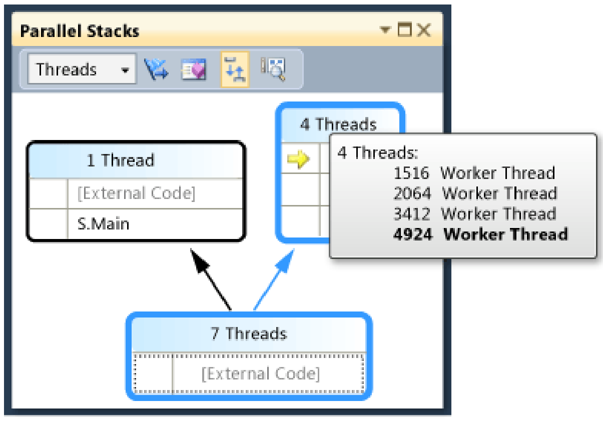
1. On the **Debug** menu, point to **Windows** and then click **Parallel Stacks**. Make sure that **Threads** is selected in the box at the upper-left corner.

   By using the **Parallel Stacks** window, you can view multiple call stacks at the same time in one view. The following illustration shows the **Parallel Stacks** window above the **Call Stack** window.

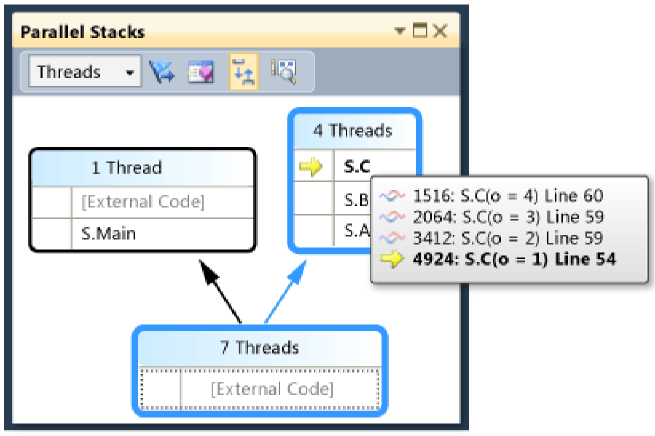


	Name	Language
➪	S.C(o=1) Line 54	C#
	S.B(o=1) Line 39	C#
	S.A(o=1) Line 35	C#
	[External Code]	

   The call stack of the Main thread appears in one box and the call stacks for the other four threads are grouped in another box. Four threads are grouped together because their stack frames share the same method contexts; that is, they are in the same methods: `A`, `B`, and `C`. To view the thread IDs and names of the threads that share the same box, hover over the header (**4 Threads**). The current thread is displayed in bold, as shown in the following illustration.



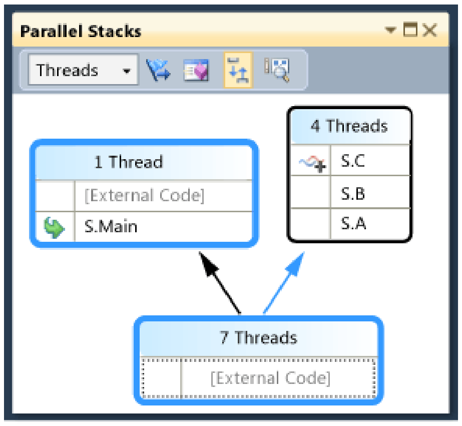
   The yellow arrow indicates the active stack frame of the current thread. To get more information, hover over it



1516: S.C(o = 4) Line 60
2064: S.C(o = 3) Line 59
3412: S.C(o = 2) Line 59
4924: S.C(o = 1) Line 54

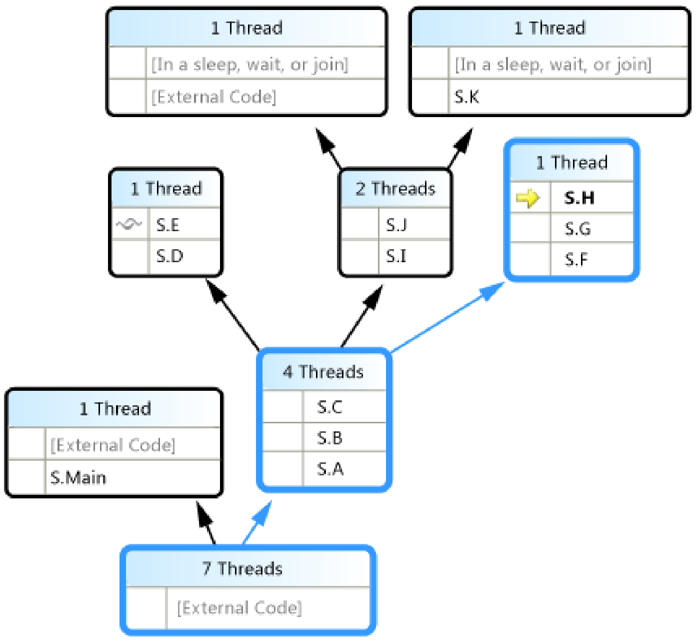
You can set how much detail to show for the stack frames (**Module Names**, **Parameter Types**, **Parameter Names**, **Parameter Values**, **Line Numbers** and **Byte Offsets**) by right-clicking in the **Call Stack** window.

A blue highlight around a box indicates that the current thread is part of that box. The current thread is also indicated by the bold stack frame in the tooltip. If you double-click the Main thread in the Threads window, you can observe that the blue highlight in the **Parallel Stacks** window moves accordingly.



To resume execution until the second breakpoint

1. To resume execution until the second breakpoint is hit, on the **Debug** menu, click **Continue**. The following illustration shows the thread tree at the second breakpoint.



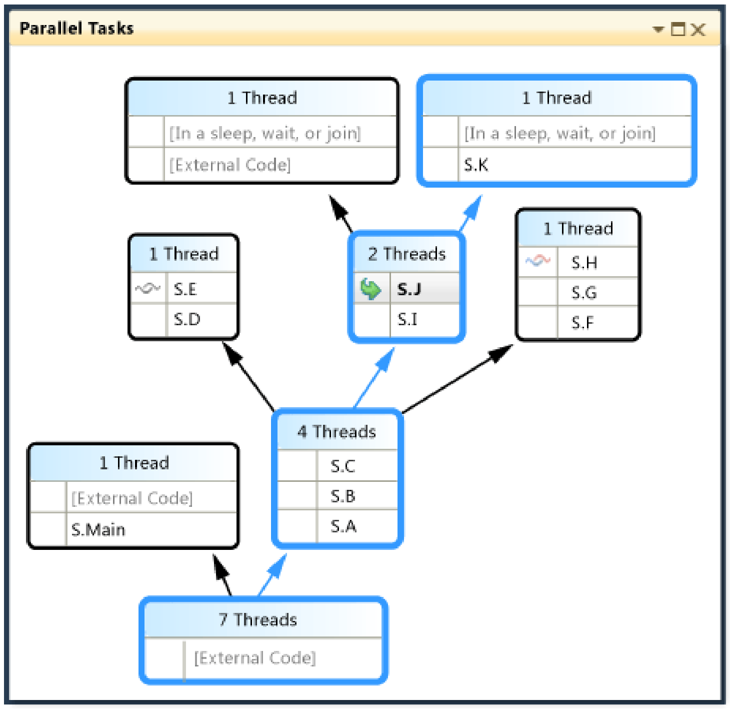
At the first breakpoint, four threads all went from S.A to S.B to S.C methods. That information is still visible in the **Parallel Stacks** window, but the four threads have progressed further. One of them continued to S.D and then S.E. Another continued to S.F, S.G, and S.H. Two others continued to S.I and S.J, and from there one of them went to S.K and the other continued to non-user External Code.

You can hover over the box header, for example, **1 Thread** or **2 Threads**, to see the thread IDs of the threads. You can hover over stack frames to see thread IDs plus other frame details. The blue highlight indicates the current thread and the yellow arrow indicates the active stack frame of the current thread.

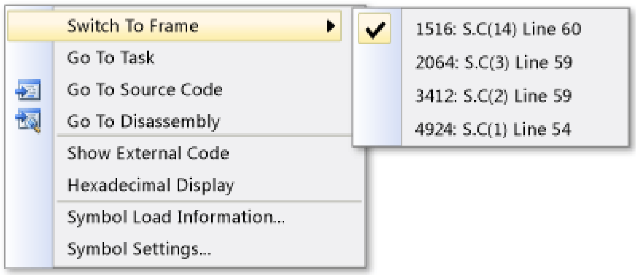
The cloth-threads icon (overlapping blue and red waved lines) indicate the active stack frames of the noncurrent threads. In the **Call Stack** window, double-click S.B to switch frames. The **Parallel Stacks** window indicates the current stack frame of the current thread by using a green curved arrow icon.

In the **Threads** window, switch between threads and observe that the view in the **Parallel Stacks** window is updated.

You can switch to another thread, or to another frame of another thread, by using the shortcut menu in the **Parallel Stacks** window. For example, right-click S.J, point to **Switch To Frame**, and then click a command.

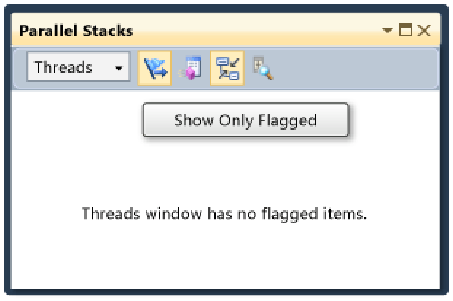


Right-click S.C and point to **Switch To Frame**. One of the commands has a check mark that indicates the stack frame of the current thread. You can switch to that frame of the same thread (just the green arrow will move) or you can switch to the other thread (the blue highlight will also move). The following illustration shows the submenu.



When a method context is associated with just one stack frame, the box header displays **1 Thread** and you can switch to it by double-clicking. If you double-click a method context that has more than 1 frame associated with it, then the menu automatically pops up. As you hover over the method contexts, notice the black triangle at the right. Clicking that triangle also displays the shortcut menu.

For large applications that have many threads, you may want to focus on just a subset of threads. The **Parallel Stacks** window can display call stacks only for flagged threads. On the toolbar, click the **Show Only Flagged** button next to the list box.



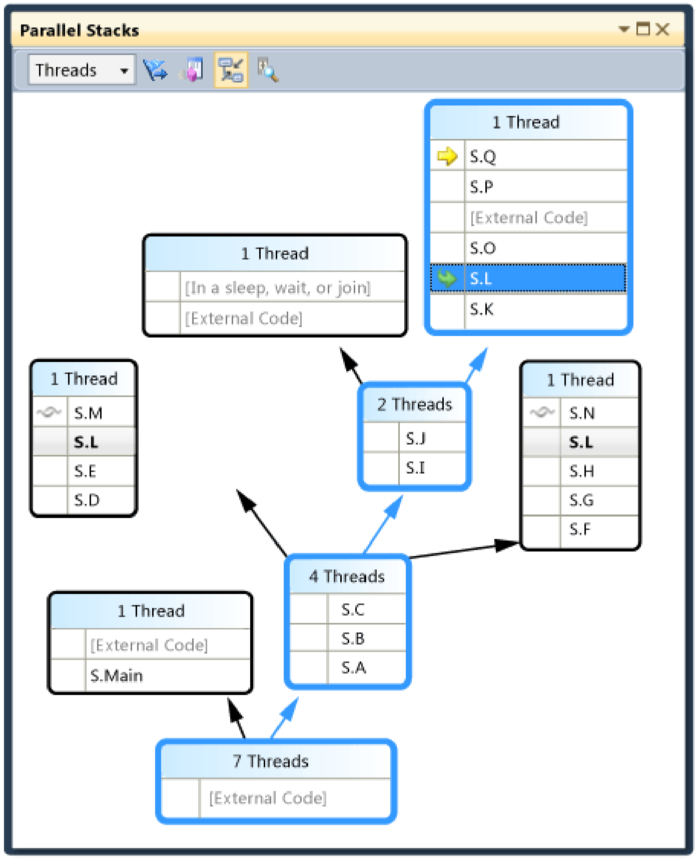
Threads window has no flagged items.

Next, in the **Threads** window, flag threads one by one to see how their call stacks appear in the **Parallel Stacks** window. To flag threads, use the shortcut menu or the first cell of a thread. Click the **Show Only Flagged** toolbar button again to show all threads.

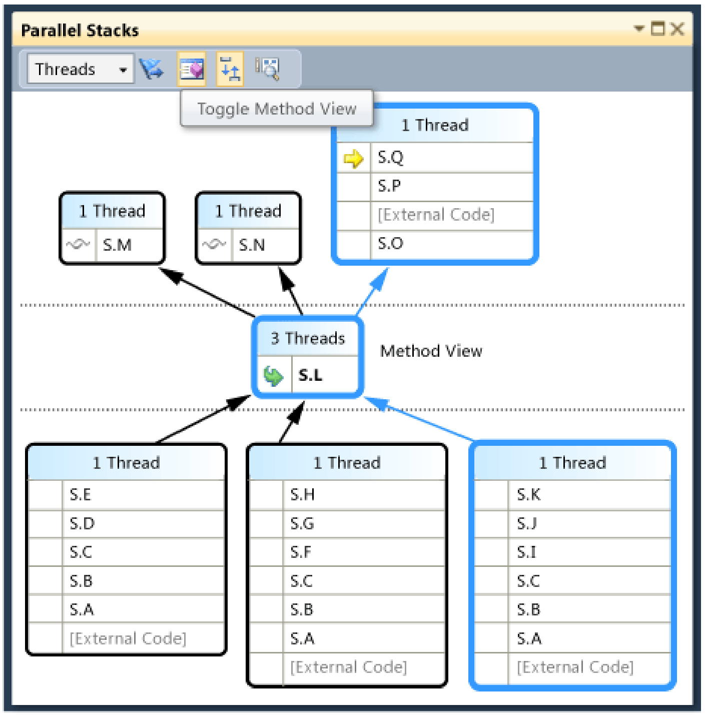
To resume execution until the third breakpoint

1. To resume execution until the third breakpoint is hit, on the **Debug** menu, click **Continue**.

   When multiple threads are in the same method but the method was not at the beginning of the call stack, the method appears in different boxes. An example at the current breakpoint is S.L, which has three threads in it and appears in three boxes. Double-click S.L.



Notice that S.L is bold in the other two boxes so that you can see where else it appears. If you want to see which frames call into S.L and which frames it calls, click the **Toggle Method View** button on the toolbar. The following illustration shows the method view of The **Parallel Stacks** window.

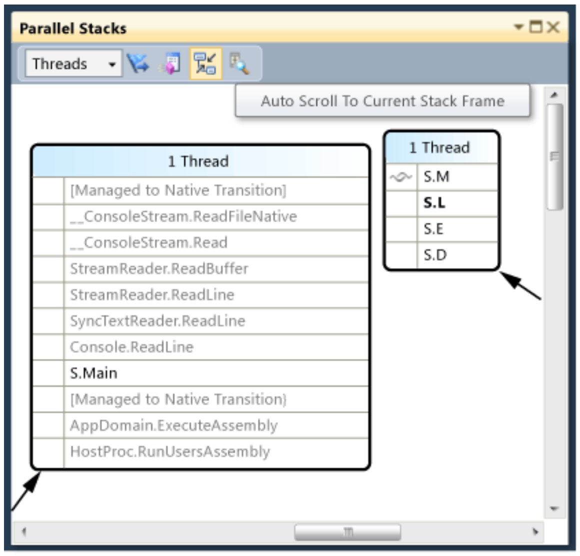


Notice how the diagram pivoted on the selected method and positioned it in its own box in the middle of the view. The callees and callers appear on the top and bottom. Click the **Toggle Method View** button again to leave this mode.

The shortcut menu of the **Parallel Stacks** window also has the following other items.

- **Hexadecimal Display** toggles the numbers in the tooltips between decimal and hexadecimal.
- **Symbol Load Information** and **Symbol Settings** open the respective dialog boxes.
- **Go To Source Code** and **Go To Disassembly** navigate in the editor to the selected method.
- **Show External Code** displays all the frames even if they are not in user code. Try it to see the diagram expand to accommodate the additional frames (which may be dimmed because you do not have symbols for them).

When you have large diagrams and you step to the next breakpoint, you may want the view to auto scroll to the active stack frame of the current thread; that is, the thread that hit the breakpoint first. In the **Parallel Stacks** window, make sure that the **Auto Scroll to Current Stack Frame** button on the toolbar is on.


Parallel Stacks
Threads
Auto Scroll To Current Stack Frame
1 Thread
[Managed to Native Transition]
__ConsoleStream.ReadFileNative
__ConsoleStream.Read
StreamReader.ReadBuffer
StreamReader.ReadLine
SyncTextReader.ReadLine
Console.ReadLine
S.Main
[Managed to Native Transition]
AppDomain.ExecuteAssembly
HostProc.RunUsersAssembly
1 Thread
S.M
S.L
S.E
S.D

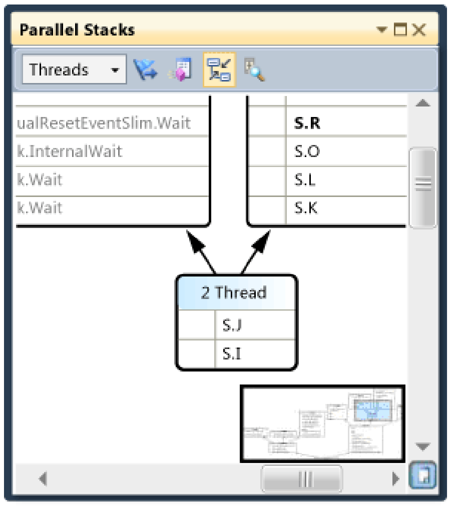
2. Before you continue, in the **Parallel Stacks** window, scroll all the way to the left and all the way down.

To resume execution until the fourth breakpoint

1. To resume execution until the fourth breakpoint is hit, on the **Debug** menu, click **Continue**.

   Notice how the view autoscrolled into place. Switch threads in the **Threads** window or switch stack frames in the **Call Stack** window and notice how the view always autoscrolls to the correct frame. Turn off **Auto Scroll to Current Tool Frame** option and view the difference.

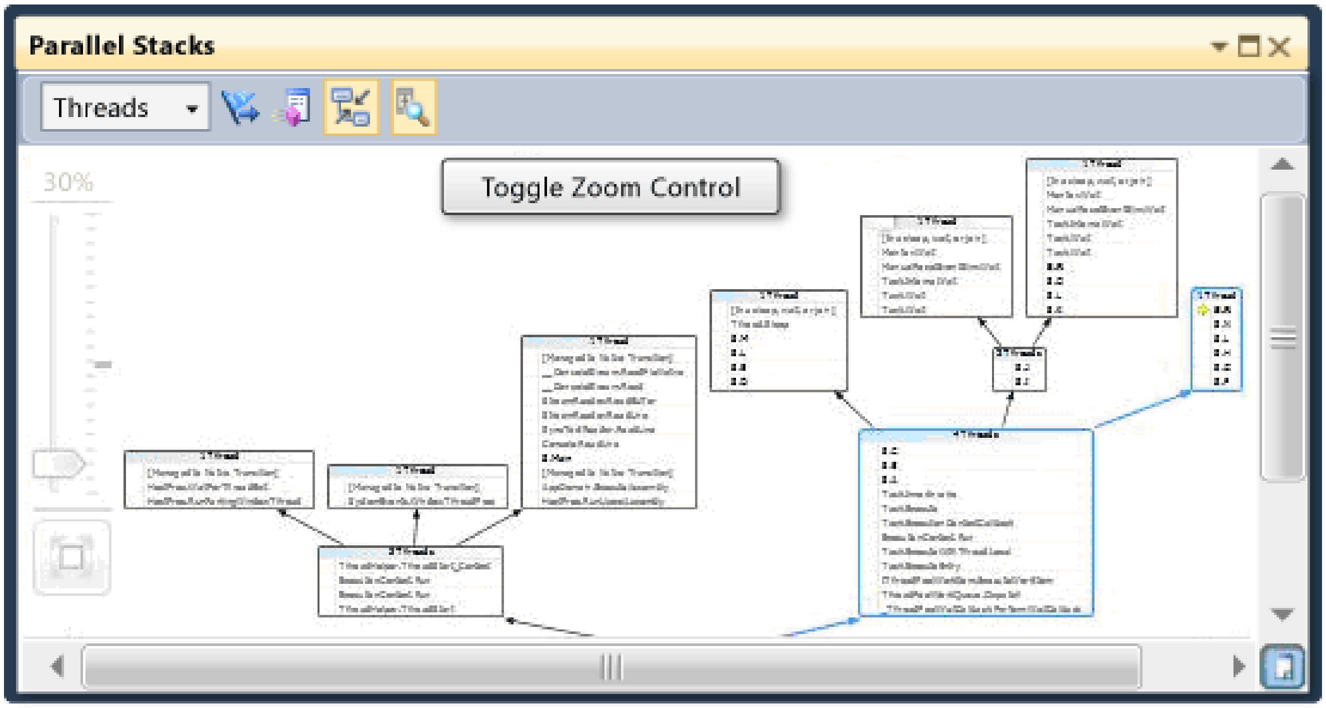
   The **Bird's Eye View** also helps with large diagrams in the **Parallel Stacks** window. You can see the **Bird's Eye View** by clicking the button between the scroll bars on the lower-right corner of the window, as shown in the following illustration.



You can move the rectangle to quickly pan around the diagram.

Another way to move the diagram in any direction is to click a blank area of the diagram and drag it where you want it.

To zoom in and out of the diagram, press and hold CTRL while you move the mouse wheel. Alternatively, click the Zoom button on the toolbar and then use the Zoom tool.



You can also view the stacks in a top-down direction instead of bottom-up, by clicking the **Tools** menu, clicking **Options**, and then select or clear the option under the **Debugging** node.

2. Before you continue, on the **Debug** menu, click **Stop Debugging** to end execution.

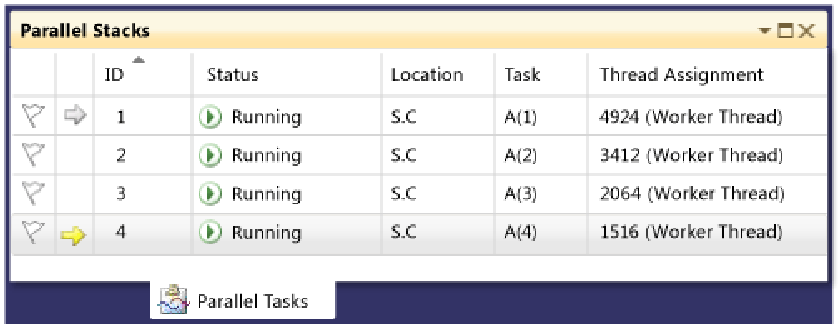
Using the Parallel Tasks Window and the Tasks View of the Parallel Stacks window

We recommended that you complete the earlier procedures before you continue.

To restart the application until the first breakpoint is hit

1. On the **Debug** menu, click **Start Debugging** and wait for the first breakpoint to be hit.
2. On the **Debug** menu, point to **Windows** and then click **Threads**. Dock the **Threads** window at the bottom of Visual Studio.
3. On the **Debug** menu, point to **Windows** and click **Call Stack**. Dock the **Call Stack** window at the bottom Visual Studio.

4. Double-click a thread in the **Threads** window to makes it current. Current threads have the yellow arrow. When you change the current thread, the other windows are updated. Next, we will examine tasks.
5. On the **Debug** menu, point to **Windows** and then click **Parallel Tasks**. The following illustration shows the **Parallel Tasks** window.



		ID	Status	Location	Task	Thread Assignment
		1	Running	S.C	A(1)	4924 (Worker Thread)
		2	Running	S.C	A(2)	3412 (Worker Thread)
		3	Running	S.C	A(3)	2064 (Worker Thread)
		4	Running	S.C	A(4)	1516 (Worker Thread)

For each running Task, you can read its ID, which is returned by the same-named property, the ID and name of the thread that runs it, its location (hovering over that displays a tooltip that has the whole call stack). Also, under the **Task** column, you can see the method that was passed into the task; in other words, the starting point.

You can sort any column. Notice the sort glyph that indicates the sort column and direction. You can also reorder the columns by dragging them left or right.

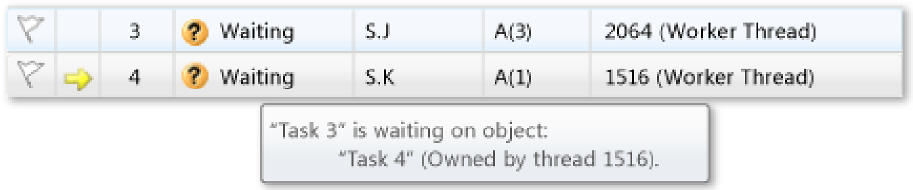
The yellow arrow indicates the current task. You can switch tasks by double-clicking a task or by using the shortcut menu. When you switch tasks, the underlying thread becomes current and the other windows are updated.

When you manually switch from one task to another, the yellow arrow moves, but a white arrow still shows the task that caused the debugger to break.

To resume execution until the second breakpoint

1. To resume execution until the second breakpoint is hit, on the **Debug** menu, click **Continue**.

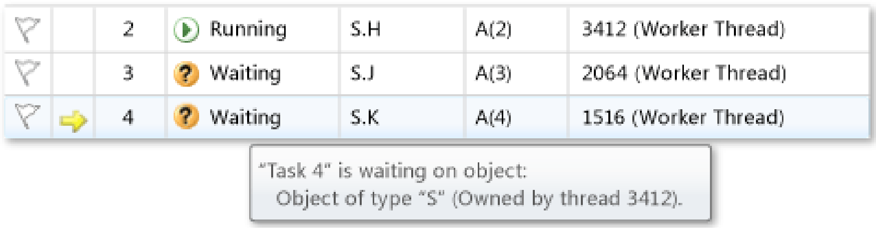
   Previously, the **Status** column showed all tasks as Running, but now two of the tasks are Waiting. Tasks can be blocked for many different reasons. In the **Status** column, hover over a waiting task to learn why it is blocked. For example, in the following illustration, task 3 is waiting on task 4.



		3	Waiting	S.J	A(3)	2064 (Worker Thread)
		4	Waiting	S.K	A(1)	1516 (Worker Thread)

"Task 3" is waiting on object:
"Task 4" (Owned by thread 1516).

Task 4, in turn, is waiting on a monitor owned by the thread assigned to task 2.



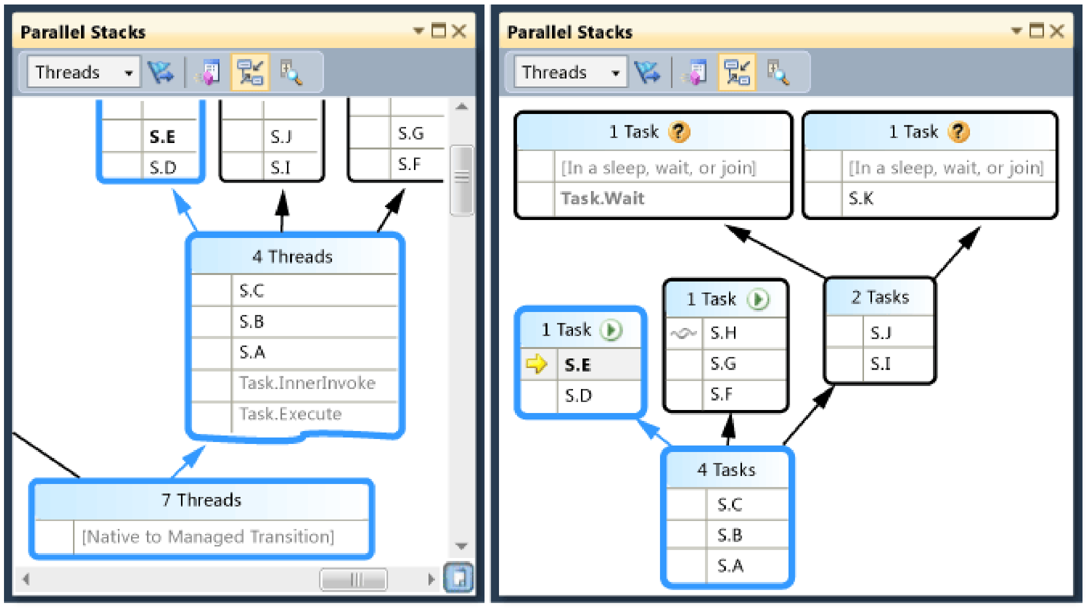
		2	Running	S.H	A(2)	3412 (Worker Thread)
		3	Waiting	S.J	A(3)	2064 (Worker Thread)
		4	Waiting	S.K	A(4)	1516 (Worker Thread)

"Task 4" is waiting on object:
Object of type "S" (Owned by thread 3412).

You can flag a task by clicking the flag in the first column of the **Parallel Tasks** window.

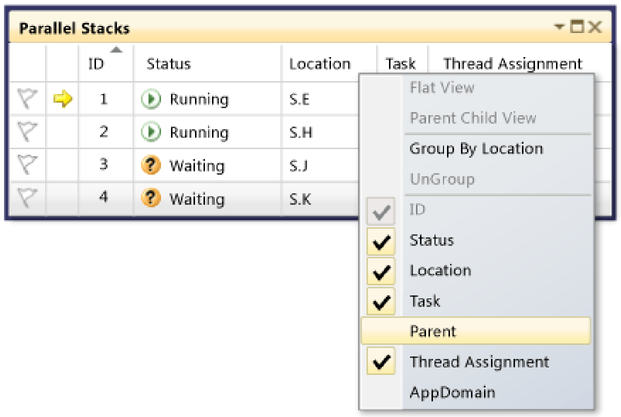
You can use flagging to track tasks between different breakpoints in the same debugging session or to filter for tasks whose call stacks are shown in the **Parallel Stacks** window.

When you used the **Parallel Stacks** window earlier, you viewed the application threads. View the **Parallel Stacks** window again, but this time view the application tasks. Do this by selecting **Tasks** in the box on the upper left. The following illustration shows the Tasks View.



Threads that are not currently executing tasks are not shown in the Tasks View of the **Parallel Stacks** window. Also, for threads that execute tasks, some of the stack frames that are not relevant to tasks are filtered from the top and bottom of the stack.

View the **Parallel Tasks** window again. Right-click any column header to see a shortcut menu for the column.



		ID	Status	Location	Task	Thread Assignment
		1	Running	S.E		
		2	Running	S.H		
		3	Waiting	S.J		
		4	Waiting	S.K		

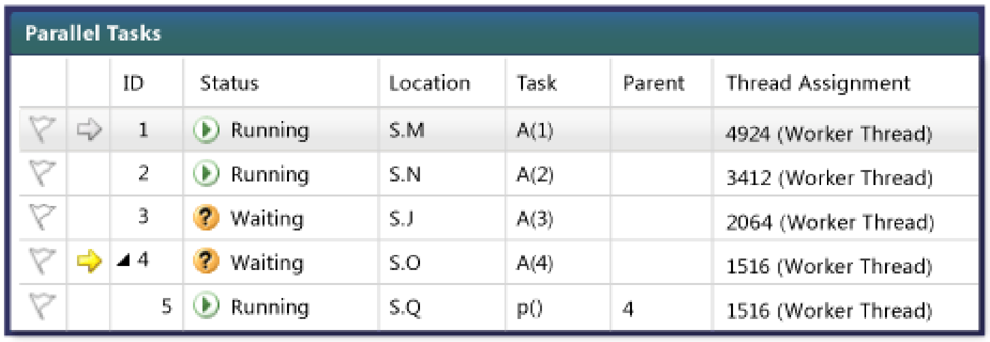
You can use the shortcut menu to add or remove columns. For example, the AppDomain column is not selected; therefore, it is not displayed in the list. Click **Parent**. The **Parent** column appears without values for any of the four tasks.

To resume execution until the third breakpoint

1. To resume execution until the third breakpoint is hit, on the **Debug** menu, click **Continue**.

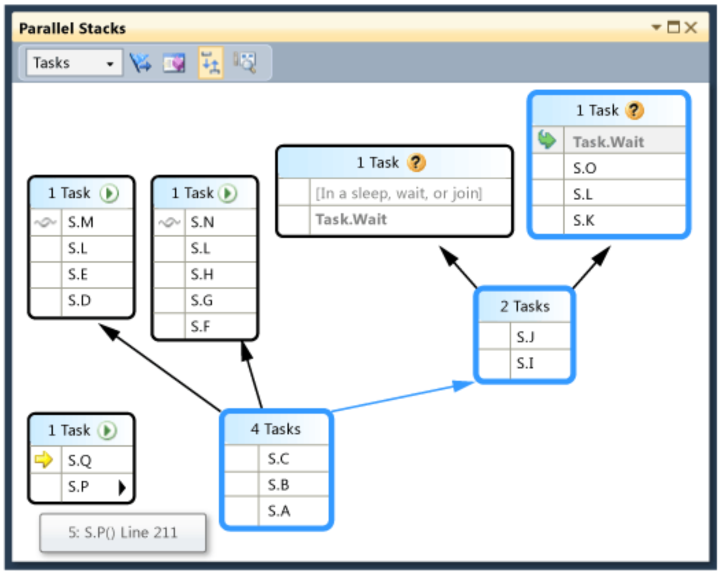
   A new task, task 5, is now running and task 4 is now waiting. You can see why by hovering over the waiting task in the **Status** window. In the **Parent** column, notice that task 4 is the parent of task 5.

   To better visualize the parent-child relationship, right-click the **Parent** column header and then click **Parent Child View**. You should see the following illustration.

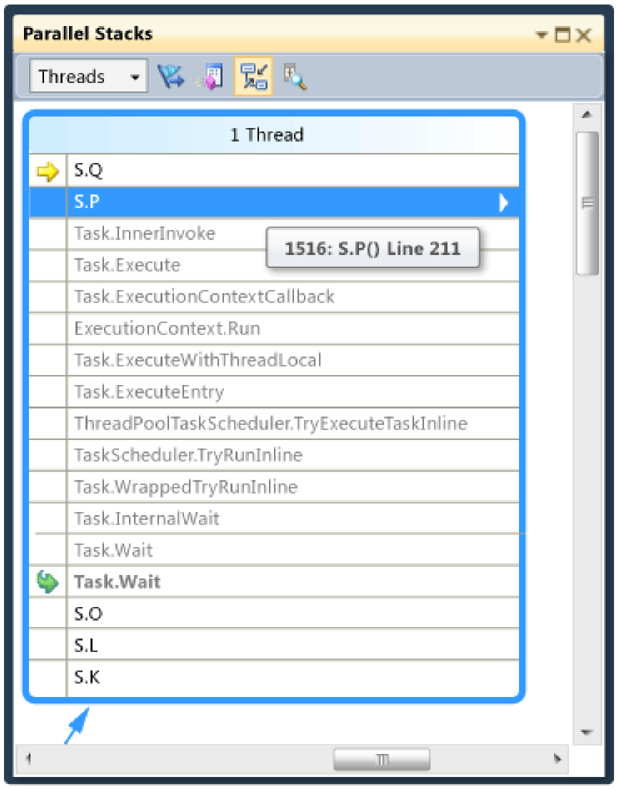


		ID	Status	Location	Task	Parent	Thread Assignment
		1	Running	S.M	A(1)		4924 (Worker Thread)
		2	Running	S.N	A(2)		3412 (Worker Thread)
		3	Waiting	S.J	A(3)		2064 (Worker Thread)
		◢ 4	Waiting	S.O	A(4)		1516 (Worker Thread)
		5	Running	S.Q	p()	4	1516 (Worker Thread)

   Notice that task 4 and task 5 are running on the same thread. This information is not displayed in the **Threads** window; seeing it here is another benefit of the **Parallel Tasks** window. To confirm this, view the **Parallel Stacks** window. Make sure that you are viewing **Tasks**. Locate tasks 4 and 5 by double-clicking them in the **Parallel Tasks** window. When you do, the blue highlight in the **Parallel Stacks** window is updated. You can also locate tasks 4 and 5 by scanning the tooltips on the **Parallel Stacks** window.



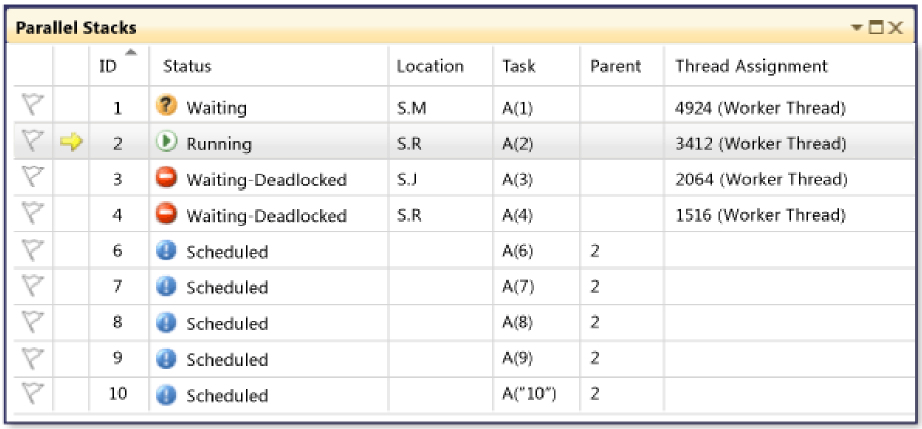
In the **Parallel Stacks** window, right-click S.P, and then click **Go To Thread**. The window switches to Threads View and the corresponding frame is in view. You can see both tasks on the same thread.



This is another benefit of the Tasks View in the **Parallel Stacks** window, compared to the **Threads** window.

To resume execution until the fourth breakpoint

1. To resume execution until the third breakpoint is hit, on the **Debug** menu, click **Continue**. Click the **ID** column header to sort by ID. You should see the following illustration.



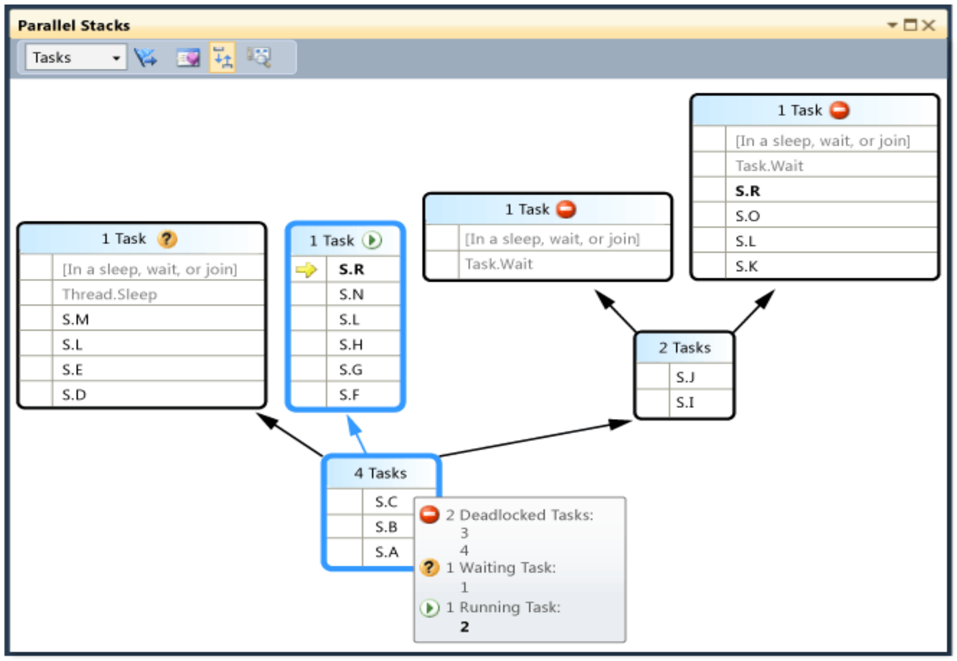
Parallel Stacks

		ID	Status	Location	Task	Parent	Thread Assignment
		1	Waiting	S.M	A(1)		4924 (Worker Thread)
	➡	2	Running	S.R	A(2)		3412 (Worker Thread)
		3	Waiting-Deadlocked	S.J	A(3)		2064 (Worker Thread)
		4	Waiting-Deadlocked	S.R	A(4)		1516 (Worker Thread)
		6	Scheduled		A(6)	2	
		7	Scheduled		A(7)	2	
		8	Scheduled		A(8)	2	
		9	Scheduled		A(9)	2	
		10	Scheduled		A("10")	2	

Because task 5 has completed, it is no longer displayed. If that is not the case on your computer and the deadlock is not shown, step one time by pressing F11.

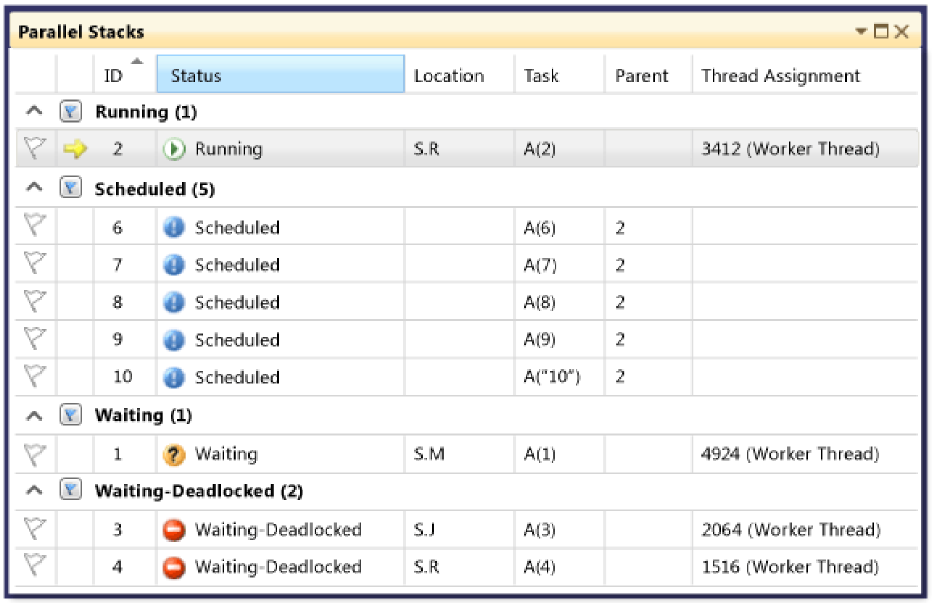
Task 3 and task 4 are now waiting on each other and are deadlocked. There are also 5 new tasks that are children of task 2 and are now scheduled. Scheduled tasks are tasks that have been started in code but have not run yet. Therefore, their **Location** and **Thread Assignment** columns are empty.

View the **Parallel Stacks** window again. The header of each box has a tooltip that shows the thread IDs and names. Switch to Tasks View in the **Parallel Stacks** window. Hover over a header to see the task ID and name, and the status of the task, as shown in the following illustration.



2 Deadlocked Tasks:
3
4
1 Waiting Task:
1
1 Running Task:
2

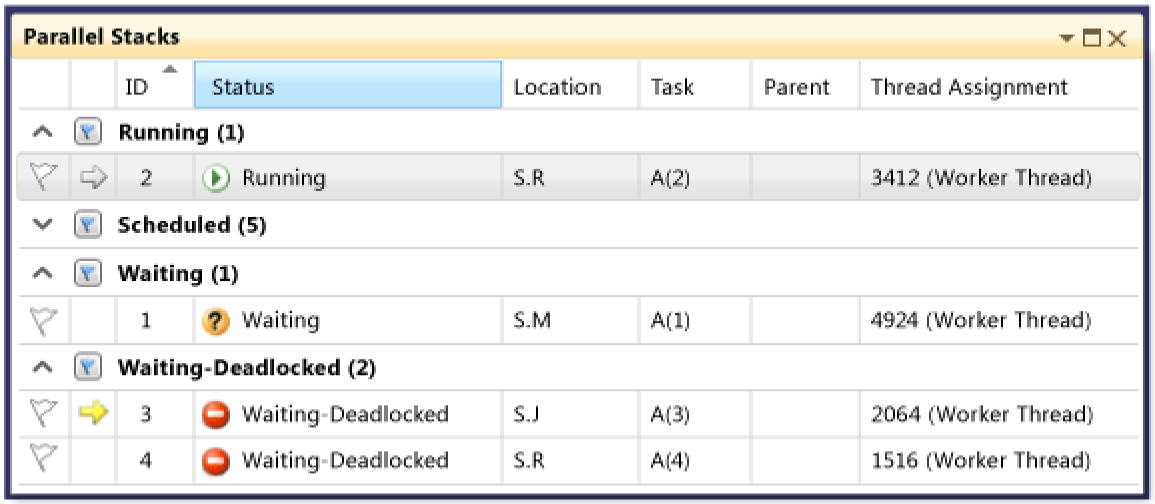
You can group the tasks by column. In the **Parallel Tasks** window, right-click the **Status** column header and then click **Group by Status**. The following illustration shows the **Parallel Tasks** window grouped by status.



Parallel Stacks

		ID	Status	Location	Task	Parent	Thread Assignment
Running (1)							
		2	Running	S.R	A(2)		3412 (Worker Thread)
Scheduled (5)							
		6	Scheduled		A(6)	2	
		7	Scheduled		A(7)	2	
		8	Scheduled		A(8)	2	
		9	Scheduled		A(9)	2	
		10	Scheduled		A("10")	2	
Waiting (1)							
		1	Waiting	S.M	A(1)		4924 (Worker Thread)
Waiting-Deadlocked (2)							
		3	Waiting-Deadlocked	S.J	A(3)		2064 (Worker Thread)
		4	Waiting-Deadlocked	S.R	A(4)		1516 (Worker Thread)

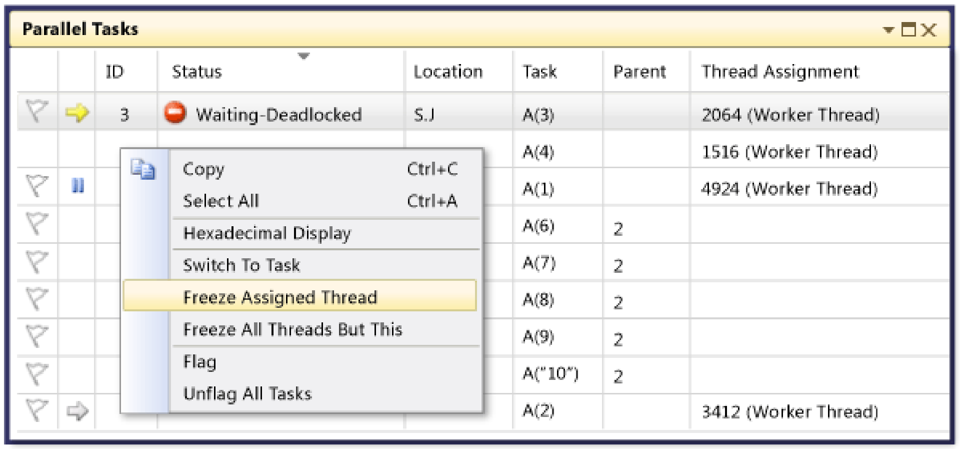
You can also group by any other column. By grouping tasks, you can focus on a subset of tasks. Each collapsible group has a count of the items that are grouped together. You can also quickly flag all items in the group by clicking the **Flag** button to the right of the **Collapse** button.



Parallel Stacks

		ID	Status	Location	Task	Parent	Thread Assignment
Running (1)							
		2	Running	S.R	A(2)		3412 (Worker Thread)
Scheduled (5)							
Waiting (1)							
		1	Waiting	S.M	A(1)		4924 (Worker Thread)
Waiting-Deadlocked (2)							
		3	Waiting-Deadlocked	S.J	A(3)		2064 (Worker Thread)
		4	Waiting-Deadlocked	S.R	A(4)		1516 (Worker Thread)

The last feature of the **Parallel Tasks** window to examine is the shortcut menu that is displayed when you right-click a task.



		ID	Status	Location	Task	Parent	Thread Assignment
		3	Waiting-Deadlocked	S.J	A(3)		2064 (Worker Thread)
					A(4)		1516 (Worker Thread)
					A(1)		4924 (Worker Thread)
					A(6)	2	
					A(7)	2	
					A(8)	2	
					A(9)	2	
					A("10")	2	
					A(2)		3412 (Worker Thread)

The shortcut menu displays different commands, depending on the status of the task. The commands may include **Copy**, **Select All**, **Hexadecimal Display**, **Switch to Task**, **Freeze Assigned Thread**, **Freeze All Threads But This**, and **Thaw Assigned Thread**, and **Flag**.

You can freeze the underlying thread of a task, or tasks, or you can freeze all threads except the assigned one. A frozen thread is represented in the **Parallel Tasks** window as it is in the **Threads** window, by a blue *pause* icon.

Summary

This walkthrough demonstrated the **Parallel Tasks** and **Parallel Stacks** debugger windows. Use these windows on real projects that use multithreaded code. You can examine parallel code written in C++, C#, or Visual Basic.