# MANAGING EXECUTION STATE OF COMPONENTS IN A GRAPH-BASED PROGRAM SPECIFICATION FOR CONTROLLING THEIR ASSOCIATED TASKS

## CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims priority to U.S. Application Serial No. 62/044,684, filed on September 2, 2014, and U.S. Application Serial No. 62/164,175, filed on May 20, 2015.

## BACKGROUND

This description relates to an approach to managing state for controlling tasks.

One approach to data flow computation makes use of a graph-based representation in which computational components corresponding to nodes (vertices) of a graph are coupled by data flows corresponding to links (directed edges) of the graph (called a "dataflow graph"). A downstream component connected to an upstream component by a data flow link receives an ordered stream of input data elements, and processes the input data elements in the received order, optionally generating one or more corresponding flows of output data elements. A system for executing such graph-based computations is described in prior U.S. Patent 5,966,072, titled "EXECUTING COMPUTATIONS EXPRESSED AS GRAPHS," incorporated herein by reference. In an implementation related to the approach described in that prior patent, each component is implemented as a process that is hosted on one of typically multiple computer servers. Each computer server may have multiple such component processes active at any one time, and an operating system (e.g., Unix) scheduler shares resources (e.g., processor time, and/or processor cores) among the components hosted on that server. In such an implementation, data flows between components may be implemented using data communication services of the operating system and data network connecting the servers (e.g., named pipes, TCP/IP sessions, etc.). A subset of the components generally serve as sources and/or sinks of data from the overall computation, for example, to and/or from data files, database tables, and external data flows. After the component processes and data flows are established, for example, by a coordinating process, data then flows through the overall computation system implementing the computation expressed as a graph generally governed by availability of input data at each component and scheduling

What is claimed is:

1. A method for controlling execution of components, each of said components including control code for executing a task associated with said component, the method including:

receiving information representative of a graph-based program specification having components, each of which corresponds to at least one task, and directed links between ports of said components, said directed links defining a dependency between said components,

wherein said components include a first component and a second component,

wherein a directed link exists between a port of said first component and a port of said second component,

wherein said first component specifies first-component execution code that when compiled enables execution of a first task,

wherein said second component specifies second-component execution code that when compiled enables execution of a second task, and

compiling said graph-based program specification, wherein compiling said graph-based program specification includes grafting first control code to said first-component execution code,

wherein said first control code changes a state of said second component to a state in a group of states that includes a pending state, an active state, and a suppressed state, and

said first control code includes code that causes invoking said second component if said state of said second component changes from said pending state to said active state, and said first control code includes code that causes suppressing said second component if said state of said second component changes from said pending state to said suppressed state.

2. The method of claim 1, wherein compiling said graph-based program specification further includes grafting second control code to said second-component execution code, and in response to invocation of said second component, said second control code causes said second-component execution code to execute said second task.

3. The method of claim 1, wherein said first control code causes invocation of said second component.

4. The method of claim 3, wherein invocation of said second task causes allocation of computational resources to said second task.

5. The method of claim 3, wherein invocation of said second task causes said second task to transition from consuming no computational resources to consuming computational resources.

6. The method of claim 1, wherein said first control code changes said state of said second component by changing a state of an accumulator indicating that at least one input to said second component is ready.

7. The method of claim 1, wherein said first control code causes suppression of said second component.

8. The method of claim 7, wherein said state of said second component includes a state of a flag set to a value indicating said suppression.

9. The method of claim 1, wherein compiling said graph-based program specification further includes generating at least one control graph for at least one subset of components of the graph-based program specification including said first component and said second component, wherein the control graph includes components of the graph-based program specification and connections between components based on links in the graph-based program specification between ports that enforce serial execution.

10. The method of claim 9, wherein grafting said first control code includes determining said first control code according to said connections of said control graph.

11. The method of claim 9, wherein generating said control graph includes: adding a begin component and a finish component and connecting one or more components of the graph-based program specification to at least one of said begin component and finish component according to a topology of the graph-based program specification.

12. The method of claim 1, wherein said first control code includes an epilogue grafted at an end of said first-component execution code for execution after said first-component execution code.

13. The method of claim 1, wherein compiling said graph-based program specification further includes grafting second control code to said second-component execution code, and said second control code includes a prologue grafted at a beginning of said second-component execution code for execution before said second-component execution code.

14. The method of claim 13, wherein said second control code monitors the state of said second

component changed by said first control code.

15. The method of claim 1, wherein said group of states also includes a complete state, and said second component changes from said active state to said complete state after execution of said second task has completed.

16. The method of claim 15, wherein said first control code changes said state of said second component in response to said first component changing from said active state to said complete state.

17. The method of claim 9, wherein said subset of components of the graph-based program specification including said first component and said second component includes at least one link between ports of first linked components, wherein said ports enforce serial execution of said first linked components, and at least one link between second linked components that allow concurrent execution of said second linked components.

18. Software stored in a non-transitory form on a computer-readable medium, for controlling execution of components, each of said components including control code for executing a task associated with said component, the software including instructions for causing a computing system to:

receive information representative of a graph-based program specification having components, each of which corresponds to at least one task, and directed links between ports of said components, said directed links defining a dependency between said components,

wherein said components include a first component and a second component,

wherein a directed link exists between a port of said first component and a port of said second component,

wherein said first component specifies first-component execution code that when compiled enables execution of a first task,

wherein said second component specifies second-component execution code that when compiled enables execution of a second task, and

compile said graph-based program specification, wherein compiling said graph-based program specification includes grafting first control code to said first-component execution code,

wherein said first control code changes a state of said second component to a state in a group of states that includes a pending state, an active state, and a suppressed state, and

said first control code includes code that causes invoking said second component if said state of said second component changes from said pending state to said active state, and said first control code includes code that causes suppressing said second component if said state of said second component changes from said pending state to said suppressed state.

19. The software of claim 18, wherein compiling said graph-based program specification further includes grafting second control code to said second-component execution code, and in response to invocation of said second component, said second control code causes said second-component execution code to execute said second task.

20. The software of claim 18, wherein said first control code changes said state of said second component by changing a state of an accumulator indicating that at least one input to said second component is ready.

21. The software of claim 18, wherein compiling said graph-based program specification further includes generating at least one control graph for at least one subset of components of the graph-based program specification including said first component and said second component, wherein the control graph includes components of the graph-based program specification and connections between components based on links in the graph-based program specification between ports that enforce serial execution.

22. The software of claim 21, wherein grafting said first control code includes determining said first control code according to said connections of said control graph.

23. The software of claim 21, wherein generating said control graph includes: adding a begin component and a finish component and connecting one or more components of the graph-based program specification to at least one of said begin component and finish component according to a topology of the graph-based program specification.

24. The software of claim 21, wherein said subset of components of the graph-based program specification including said first component and said second component includes at least one link between ports of first linked components, wherein said ports enforce serial execution of said first linked components, and at least one link between second linked components that allow concurrent execution of said second linked components.

25. The software of claim 18, wherein compiling said graph-based program specification further includes grafting second control code to said second-component execution code, and said second control code includes a prologue grafted at a beginning of said second-component execution code for execution before said second-component execution code.

26. The software of claim 25, wherein said second control code monitors the state of said second component changed by said first control code.

27. The software of claim 18, wherein said group of states also includes a complete state, and said second component changes from said active state to said complete state after execution of said second task has completed.

28. The software of claim 27, wherein said first control code changes said state of said second component in response to said first component changing from said active state to said complete state.

29. A computing system for controlling execution of components, each of said components including control code for executing a task associated with said component, the computing system including:

an input device or port configured to receive information representative of a graph-based program specification having components, each of which corresponds to at least one task, and directed links between ports of said components, said directed links defining a dependency between said components,

wherein said components include a first component and a second component,

wherein a directed link exists between a port of said first component and a port of said second component,

wherein said first component specifies first-component execution code that when compiled enables execution of a first task,

wherein said second component specifies second-component execution code that when compiled enables execution of a second task, and

at least one processor configured to compile said graph-based program specification,

wherein compiling said graph-based program specification includes grafting first control code to said first-component execution code,

wherein said first control code changes a state of said second component to a state in a group of states that includes a pending state, an active state, and a suppressed state, and

said first control code includes code that causes invoking said second component if said state of said second component changes from said pending state to said active state, and said first control code includes code that causes suppressing said second component if said state of said second component changes from said pending state to said suppressed state.

30. The computing system of claim 29, wherein compiling said graph-based program specification further includes grafting second control code to said second-component execution code, and in response to invocation of said second component, said second control code causes said second-component execution code to execute said second task.

31. The computing system of claim 29, wherein said first control code changes said state of said second component by changing a state of an accumulator indicating that at least one input to said second component is ready.

32. The computing system of claim 29, wherein compiling said graph-based program specification further

includes generating at least one control graph for at least one subset of components of the graph-based program specification including said first component and said second component, wherein the control graph includes components of the graph-based program specification and connections between components based on links in the graph-based program specification between ports that enforce serial execution.

33. The computing system of claim 32, wherein grafting said first control code includes determining said first control code according to said connections of said control graph.

34. The computing system of claim 32, wherein generating said control graph includes: adding a begin component and a finish component and connecting one or more components of the graph-based program specification to at least one of said begin component and finish component according to a topology of the graph-based program specification.

35. The computing system of claim 32, wherein said subset of components of the graph-based program specification including said first component and said second component includes at least one link between ports of first linked components, wherein said ports enforce serial execution of said first linked components, and at least one link between second linked components that allow concurrent execution of said second linked components.

36. The computing system of claim 29, wherein compiling said graph-based program specification further includes grafting second control code to said second-component execution code, and said second control code includes a prologue grafted at a beginning of said second-component execution code for execution before said second-component execution code.

37. The computing system of claim 36, wherein said second control code monitors the state of said second component changed by said first control code.

38. The computing system of claim 29, wherein said group of states also includes a complete state, and said second component changes from said active state to said complete state after execution of said second task has completed.

39. The computing system of claim 38, wherein said first control code changes said state of said second component in response to said first component changing from said active state to said complete state.

40. A computing system for controlling execution of components, each of said components including control code for executing a task associated with said component, the computing system including:

means for receiving information representative of a graph-based program specification having components, each of which corresponds to at least one task, and directed links between ports of said components, said directed links defining a dependency between said components,

wherein said components include a first component and a second component,

wherein a directed link exists between a port of said first component and a port of said second component,

wherein said first component specifies first-component execution code that when compiled enables execution of a first task,

wherein said second component specifies second-component execution code that when compiled enables execution of a second task, and

means for compiling said graph-based program specification, wherein compiling said graph-based program specification includes grafting first control code to said first-component execution code,

wherein said first control code changes a state of said second component to a state in a group of states that includes a pending state, an active state, and a suppressed state, and

said first control code includes code that causes invoking said second component if said state of said second component changes from said pending state to said active state, and said first control code includes code that causes suppressing said second component if said state of said second component changes from said pending state to said suppressed state.