US 20120222038A1

(19) **United States**

(12) **Patent Application Publication**

KATRAGADDA et al.

(10) **Pub. No.: US 2012/0222038 A1**

(43) **Pub. Date: Aug. 30, 2012**

(54) **TASK DEFINITION FOR SPECIFYING RESOURCE REQUIREMENTS**

(75) Inventors: **Ramana KATRAGADDA**, Carlsbad, CA (US); **Paul SPOLTORE**, Fremont, CA (US); **Ric HOWARD**, San Jose, CA (US)

(73) Assignee: **QST HOLDINGS, LLC**, Palo Alto, CA (US)

(21) Appl. No.: **13/336,953**

(22) Filed: **Dec. 23, 2011**

Related U.S. Application Data

(63) Continuation of application No. 10/233,175, filed on Aug. 29, 2002, now Pat. No. 8,108,656.

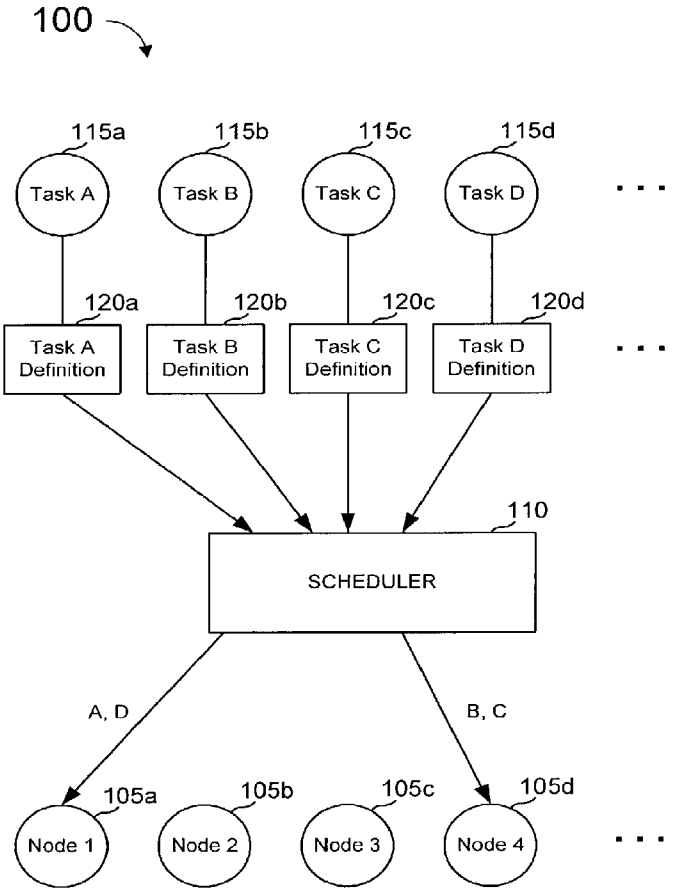
Publication Classification

(51) **Int. Cl.**
G06F 9/50 (2006.01)

(52) **U.S. Cl.** **718/104**

(57) **ABSTRACT**

Task definitions are used by a task scheduler and prioritizer to allocate task operations to a plurality of processing units. The task definition is an electronic record that specifies resources needed by, and other characteristics of, a task to be executed. Resources include types of processing nodes desired to execute the task, needed amount or rate of processing cycles, amount of memory capacity, number of registers, input/output ports, buffer sizes, etc. Characteristics of a task in clued maximum latency time, frequency of execution of a task, communication ports, and other characteristics. An exemplary task definition language and syntax is described that uses constructs including order of attempted scheduling operations, percentage or amount of resources desired by different operations, handling of multiple executable images or modules, overlays, port aliases and other features.



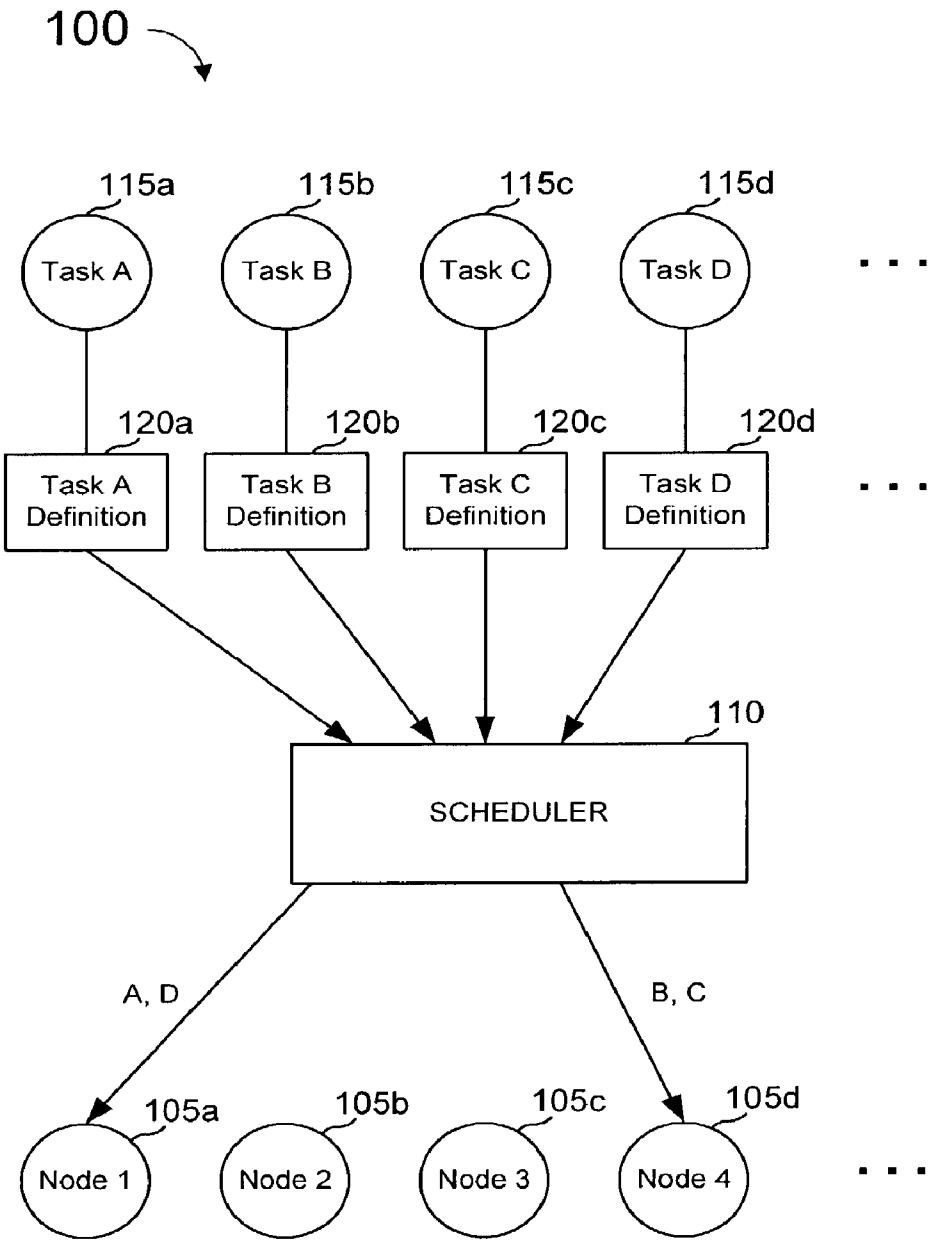

100
115a
115b
115c
115d
Task A
Task B
Task C
Task D
120a
120b
120c
120d
Task A Definition
Task B Definition
Task C Definition
Task D Definition
110
SCHEDULER
A, D
B, C
105a
105b
105c
105d
Node 1
Node 2
Node 3
Node 4

FIG. 1

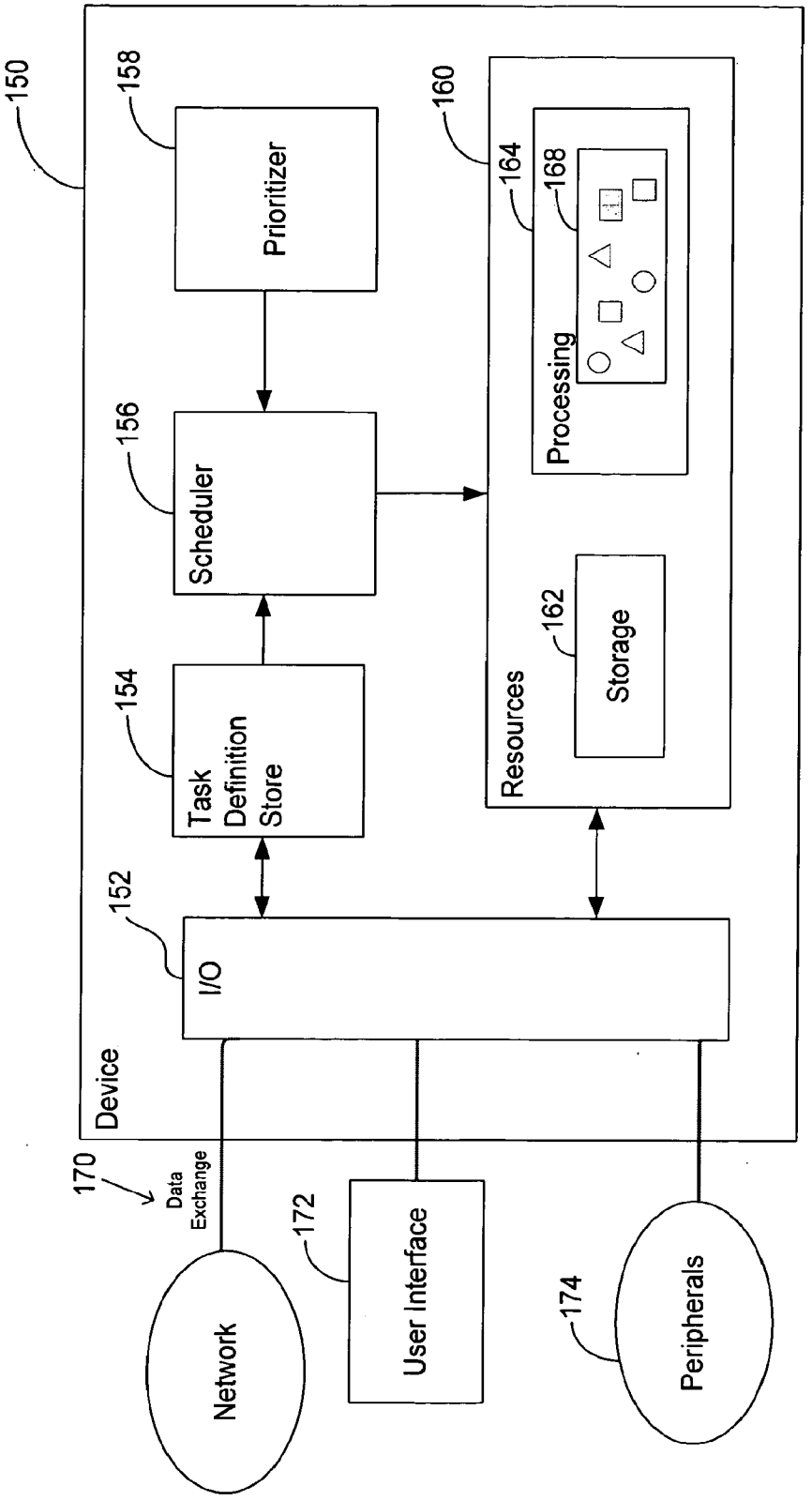

150
158
Prioritizer
160
164
168
Processing
156
Scheduler
162
Storage
Resources
154
Task Definition Store
152
I/O
Device
170
Data Exchange
172
User Interface
174
Peripherals
Network

FIG. 2

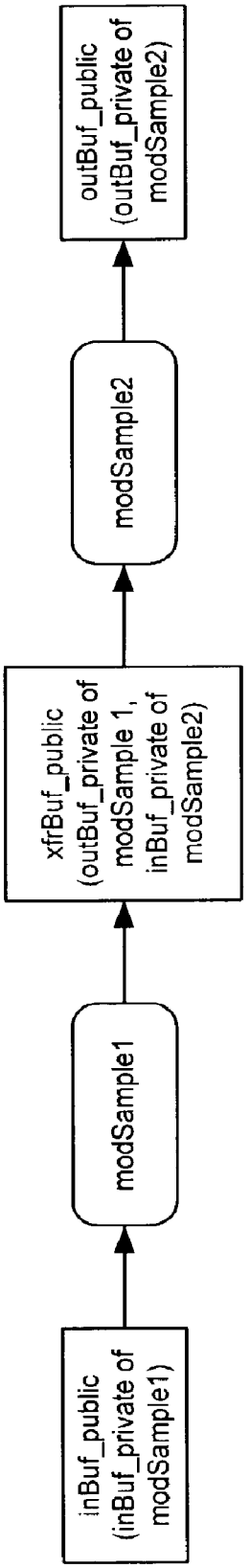

inBuf_public
(inBuf_private of
modSample1)
modSample1
xfrBuf_public
(outBuf_private of
modSample 1,
inBuf_private of
modSample2)
modSample2
outBuf_public
(outBuf_private of
modSample2)

FIG. 3

TASK DEFINITION FOR SPECIFYING RESOURCE REQUIREMENTS

CROSS-REFERENCES TO RELATED APPLICATION

[0001] This application is a continuation of U.S. patent application Ser. No. 10/233,175 filed on Aug. 29, 2002, of which the entire disclosure is incorporated herein by reference.

BACKGROUND OF THE INVENTION

[0002] This application is generally related to digital processing systems, and more specifically to digital processing systems having multiple processing nodes.

[0003] In configurable systems, it is typically desirable to be able to execute multiple tasks concurrently. If some or all of these tasks have timing constraints (for example, if "real-time" operation is desired), the configurability of the system may be limited. This can be explained with reference to typical methodologies used to implement real-time systems.

[0004] Early real-time systems were often "hand crafted" in order to meet stringent timing constraints. In particular, real-time tasks that were to be executed concurrently were analyzed to determine their detailed timing requirements. Then, a real-time operating system was "built around" these tasks such that their timing, requirements were satisfied. Because such real-time systems are tied so closely with the underlying tasks, they are not easily modifiable or extendible. For example, attempting to modify a task or add an additional task to the system could require a complete re-design of the system.

[0005] In order to make such systems "configurable," the possible configurations are typically first determined and fixed, and then the system is designed to accommodate the timing constraints of the various possible configurations. If it is desired to add a new configuration or feature, detailed knowledge of the entire system, including knowledge of tasks that might be executed at any particular time, is typically required in order to ensure that the system can execute tasks in "real-time" under the various configurations. Alternatively, the entire system might need to be re-designed. Thus, the design of the system, and that of individual tasks that are to be executed, is typically tightly controlled. This can make it difficult to add new configurations to a device, and/or to permit third-parties to develop configurations for the device.

[0006] A more flexible approach to real-time systems is often referred to as the "scheduled reservation model." Under the scheduled reservation model, the processor is viewed as a quantifiable resource that can be reserved like physical memory or disk blocks. But if two tasks require processor resources simultaneously, then one of the tasks will have to wait until the other is finished. If this task must wait too long, then it may not be able to execute in "real-time." Thus, the scheduled reservation model cannot guarantee real-time execution of all tasks.

[0007] The scheduled reservation model provides a more flexible approach to design of real-time systems. In particular, design of a task or tasks does not require detailed knowledge of the entire system and/or other tasks. Thus, unlike "hand-crafted" real-time systems, task design need not be tightly controlled, and new configurations and/or features can be developed by those (e.g., third parties, etc.) without detailed knowledge of the system or of other tasks that may run on the system. For example, new features could be developed for a configurable device without requiring any changes to the underlying system or with other tasks previously designed for the system. Further, such features could be developed by third-parties with limited knowledge of the underlying system and/or of other features. As discussed above, however, "real-time" operation might not be guaranteed.

[0008] Another approach to real-time systems is often referred to as the "fixed priority model." Under the fixed priority model, each task is assigned a priority level by developers. During operation, tasks are executed strictly based on their priority level. For example, a task with a higher priority than that of an executing task can interrupt that task, whereas a task with a lower priority than that of the executing task must wait until the executing task finishes. As with the scheduled reservation model, the fixed priority model cannot guarantee real-time execution of tasks (except for the highest priority task).

[0009] As with the scheduled reservation model, the fixed priority model provides a more flexible approach to design of real-time systems. In particular, design of a task or tasks does not require detailed knowledge of the entire system and/or other tasks. It does, however, require some knowledge of its priority vis-à-vis other tasks that may be executed by the system. Thus, task design need not be tightly controlled, but does usually require some degree of coordination. Thus, similar to systems employing a scheduled reservation model, new features for could be developed for a configurable device without requiring significant changes to the underlying system or with other tasks previously designed for the system. It may, however, require a reconfiguration of priorities of tasks that can be implemented on the device. Additionally, such features could be developed by third-parties with limited knowledge of the underlying system and/or of other features. But, "real-time" operation cannot be guaranteed.

[0010] Configurable systems having multiple processing nodes generally exacerbate the above-mentioned shortcomings and introduce others. It is desirable to provide techniques for use in configurable systems having multiple processing nodes that improve upon one or more of the above-mentioned (or other) shortcomings in the prior art.

BRIEF SUMMARY OF THE INVENTION

[0011] Task definitions are used by a task scheduler and prioritizer to allocate task operations to a plurality of processing units. The task definition is an electronic record that specifies resources needed by, and other characteristics of, a task to be executed. Resources include types of processing nodes desired to execute the task, needed amount or rate of processing cycles, amount of memory capacity, number of registers, input/output ports, buffer sizes, etc. Characteristics of a task include maximum latency time, frequency of execution of a task, communication ports, and other characteristics.

[0012] An examplary task definition language and syntax is described that uses constructs including order of attempted scheduling operations, percentage or amount of resources desired by different operations, handling of multiple executable images or modules, overlays, port aliases and other features.

[0013] In one embodiment the invention provides a computer program product comprising a computer readable storage structure embodying computer readable code therein, the computer readable code comprising a task definition code that specifies requirements of a task adapted to be executed on

a configurable device having a plurality of processing nodes, the task definition code including code that indicates processing node resources required by the task.

[0014] In another embodiment the invention provides a computer data signal embodied in a carrier wave, the computer data signal comprising a task definition code that specifies requirements of a task adapted to be executed on a configurable device having a plurality of processing nodes, the task definition code including code that indicates processing node resources required by the task.

[0015] In another embodiment the invention provides a configurable device comprising a plurality of processing nodes; a scheduler, coupled to the plurality of processing nodes, that assigns tasks to the processing nodes for execution; and a memory, coupled to the scheduler, the memory including a task definition code that specifies requirements of at least one task adapted to be executed by the configurable device, the task definition code having code that indicates processing node resources required by the task.

BRIEF DESCRIPTION OF THE DRAWINGS

[0016] FIG. **1** is a simplified block diagram illustrating an example of a system that can use embodiments of the present invention;

[0017] FIG. **2** is a simplified block diagram of an example of a device that can implement tasks that have been defined according to embodiments of the present invention; and

[0018] FIG. **3** is a simplified diagram illustrating a linking of ports and modules as specified by a task definition.

DETAILED DESCRIPTION OF THE INVENTION

Overview

[0019] Embodiments according to the present invention provide techniques for defining tasks to be implemented on a configurable device. In some specific embodiments, the target device includes a plurality of processing nodes on which tasks can be executed. In these embodiments, processing nodes can include one or more of common types of processing resources such as general purpose processors, general purpose digital signal processors, special purpose processors, finite-state machines (FSMs), application-specific integrated circuits (ASICs), etc.

[0020] FIG. **1** is a simplified block diagram illustrating an example of a system that uses embodiments of the present invention. The system **100** includes a plurality of processing nodes **105***a*, **105***b*, **105***c*, **105***d*, . . . , and a scheduler **110**. Scheduler **110** determines which of tasks **115***a*, **115***b*, **115***c*, **115***d*, . . . are to be carried out on which of the nodes **105***a*, **105***b*, **105***c*, **105***d*, ... For example, as illustrated in FIG. **1**, scheduler **110** has assigned tasks A and D for execution on node **1**, and has assigned tasks B and C for execution on node **4**.

[0021] Associated with each of tasks **115***a*, **115***b*, **115***c*, **115***d*, is a corresponding task definition (**120***a*, **120***b*, **120***c*, **120***d*, . . .). As is described subsequently, task definitions **120***a*, **120***b*, **120***c*, **120***d*, . . . provide information about their associated tasks. Such information can include a type of processing node required (or merely desired) to execute the task, required (or desired) processing resources, other required (or desired) resources (e.g., memory, buffers, ports, etc.), information for communicating with the task, etc.

[0022] Scheduler **110** can use information provided by task definitions **120***a*, **120***b*, **120***c*, **120***d*, . . . in order to assign tasks **115***a*, **115***b*, **115***c*, **115***d* to processing nodes **105***a*, **105***b*, **105***c*, **105***d*, . . . for execution in an attempt to satisfy the requirements of the tasks spelled out in the task definitions.

[0023] U.S. patent application Ser. No. 10/189,791 (Attorney Docket No. 021202-002400US), to Paul L. Spoltore, et al., entitled "Method and System for Real-Time Multitasking," filed Jul. 3, 2002 (hereinafter "Spoltore et al."), which is herein incorporated by reference in its entirety for all purposes, describes techniques for assigning tasks to processing nodes for execution. In a specific embodiment, scheduler **110** can use one or more of the techniques described in Spoltore et al. to assign **115***a*, **115***b*, **115***c*, **115***d* to processing nodes **105***a*, **105***b*, **105***c*, **105***d*, . . . for execution.

[0024] A Configurable Device

[0025] FIG. **2** is a simplified block diagram of an example of a device **150** that can implement tasks that have been defined according to embodiments of the present invention. It should be apparent, however, that aspects of the apparatus and methods described herein can be applied to many different types of computing architectures including, for example, general purpose processors, digital signal processors, custom integrated circuits, discrete circuits, etc. Additionally, aspects of the apparatus and methods described herein can be applied, in general, to any type of processing approach including, parallel processing, distributed processing, synchronous processing, asynchronous processing, etc.

[0026] Device **150** can be, for example, a consumer electronics device (or a component thereof) such as a cell phone, pager, personal digital assistant (PDA), global positioning system (GPS) receiver, etc. It should be apparent, however, that device **150** can be any type of device that can benefit from a processing engine.

[0027] Device **150** includes input/output (I/O) system **152** for providing data exchange with the external environment (illustrated at **170**), connection to peripherals **174**, and interaction with a human user via user interface **172**. Data exchange includes exchanges with digital networks such as an internet, the Internet, an intranet, an extranet, communication infrastructures such as a telephone network, radio frequency exchanges as to wireless networks, etc. Any type of physical communication or data transfer network can be employed. Any type of protocol can be used to perform the communication.

[0028] User interface **172** allows a human user to operate the device, and to perform other functions. Typically, a user interface includes a display screen and manual controls such as buttons, a pointing device (e.g., a mouse, trackball, touchpad, etc.), knobs, switches, and other types of controls. Additional output devices can include speakers, force feedback, etc. Peripherals **174** include storage devices such as disk drives, input/output devices such as keyboards, monitors, etc.

[0029] I/O system **152** can be in communication with different systems in device **150**. For example, FIG. **2** shows I/O system **152** communicating with task definition store **154** and storage and processing resources **160**. Other arrangements are possible.

[0030] Task definition store **154** is used to store programs, adaptation or configuration information, or other information used to control or manage the processing or functioning of device **150**. In one embodiment, adaptation information is used to define tasks that are executed by systems within device **150** to achieve functionality. For example, one or more

tasks might allow device **150** to communicate using time-division multiplexed access (TDMA) with a cellular phone network. One or more other tasks could provide a user with a phone directory including an interface for creating, modifying, organizing, searching, etc., the directory. Yet other tasks can implement a time-of-day clock, Internet web browsing, GPS position indication, calculator, email interface, etc. In general, any type of functionality can bc implemented by a task. Combinations of functionality can bc provided by one or more tasks. Further, a task may implement only a portion of a feature, function, or other process or functionality.

[0031] Scheduler **156** causes tasks, or portions of tasks, from task definition store **154** to be executed. Scheduler **156** can, optionally, use information provided by prioritizer **158** in determining how to specify the use of resources **160** to be used to execute a task. For example, scheduler **156** can assign all resources to a task that has been given a high priority by prioritizer **158**. Conversely, scheduler **156** may reduce resources allocated to a task, or suspend execution of a task, if the task has a low priority.

[0032] Resources **160** include storage **162** and processing resources **164**. Storage **162** can be, for example, system memory in the form of random-access memory (RAM), or other forms of storage. Storage **162** can be distributed throughout the processing elements, it can be centralized, or it can be a combination of centralized and distributed storage. Processing resources **164** can include one or more of common types of processing resources such as general purpose processors, FSMs, ASICs, etc. In one embodiment, processing resources **164** include multiple processing nodes **168** according to the adaptive computing engine ("ACE") architecture as described in U.S. patent application Ser. No. 09/815,122, entitled "Adaptive Integrated Circuitry With Heterogeneous And Reconfigurable Matrices Of Diverse And Adaptive Computational Units Having Fixed, Application Specific Computational Elements," filed Mar. 22, 2001 ("Masters"). In this embodiment, each node can be of a specific type, such as math, bit/logical, FSM, reduced-instruction set computing (RISC), etc. In this embodiment, nodes are interconnected and may have associated resources, such as memory. A detailed description of the ACE architecture is provided in Masters, which is herein incorporated by reference in its entirety for all purposes. In other embodiments, all of the nodes may be general purpose or of one type.

Task Definition

[0033] Embodiments of task definitions according to the present invention will now be described. In these embodiments, a task is comprised of one or more modules, and the requirements of each module can be specified in the task definition. It is to be understood, however, that it is not a requirement that tasks be defined in terms of modules. This is merely an example of one implementation, and one skilled in the art will recognize many modifications, equivalents, and alternatives. For example, in other embodiments, a task may not be specified in terms of modules that make up the task. In still other embodiments, a task may comprise one or more modules, where each module may in turn be comprised of one or more sub-modules. In these embodiments, a task may be specified in terms of sub-modules.

[0034] Referring to FIG. **2**, in embodiments to be used with systems such as device **150**, task definitions may be stored in task definition store **154**. In these embodiments, tasks and task definitions can be, for example, downloaded by device **150** from a remote location. For instance, with a device **150** that includes, or is coupled with, a modem, network interface, etc., a task definition could be transmitted to device **150** via a computer data signal embodied on a carrier wave, over a network such as the Internet, etc., and then loaded into task definition store **54**. Similarly, a task definition could be transmitted to a computer, and then downloaded from the computer to task definition store **154** via, for example, a serial port, parallel port, etc. In other embodiments in which device **150** includes, or is coupled with, a floppy disk drive, memory card reader, etc., task definitions could be loaded into task definition store **154** via a computer readable medium such as a disk, memory card, etc.

[0035] As described above, task definitions are associated with the tasks of which they provide information. In some embodiments, task definition may include a link, pointer, etc., to the task to which it is associated, or a location of the task in a memory, etc. In other embodiments, the task definition may be included with the task itself. For example, a task definition may be within a same file as the task itself, appended to the file, etc.

[0036] The type of information provided by task definitions will now be described. Some of this information can be used, for example, by scheduler **110** of FIG. **1**, scheduler **156** of FIG. **2**, an operating system, etc., to determine to which processing nodes tasks should be assigned or loaded, by which processing nodes tasks should be executed, etc. Additionally, some of this information can be used by an operating system, other tasks, etc., to communicate with, provide information to, etc., a task.

[0037] A. Processing Node Resources

[0038] In some embodiments, a task definition may specify resources of a processing node that are required (or desired) by the associated task. The resources could be specified, for example, in terms of a percentage of the processing node's processing power. Examples of source code for specifying processing node resources is provided subsequently.

[0039] The time between when a task can begin to execute (e.g., when data becomes available, a trigger occurs, etc.) and when the task actually begins to execute will be referred to herein as "latency." For some tasks, it may be desired that the latency not exceed (or only occasionally exceed) some maximum amount (herein referred to as the "maximum allowable latency"). Thus, in some embodiments, specifying resource requirements of a task can include specifying a maximum allowable latency. A maximum allowable latency could be specified, for example, in units of time, clock cycles, etc.

[0040] In some embodiments, specifying resource requirements of a task can include specifying a minimum amount of time required to execute the task. Time required could be specified, for example, in units of time, clock cycles, etc.

[0041] In some embodiments, specifying resource requirements of a task can include specifying a minimum frequency of execution of the task. The frequency of execution could be specified, for example, in units of time (period), clock cycles (period), hertz (frequency), etc.

[0042] Spoltore et al. describes various types of resource requirements that, in some embodiments, can be included in task definitions.

[0043] B. Processing Node Type

[0044] In some embodiments in which a device on which the task can be executed includes processing nodes of different types, a task definition may specify the type of a processing node required (or desired) by the associated task. For

example, as described with respect to FIG. **2**, device **150** can include one or more of common types of processing resources such as general purpose processors, FSMs, ASICs, etc. In one specific embodiment, device **150** includes multiple processing nodes according to the ACE architecture as described in Masters. In this embodiment, each node is of a specific type, such as math, bit/logical, FSM, or reduced-instruction set computing (RISC).

[0045] In some embodiments, a preferred choice of processing node type can be specified, as well as one or more back-up choices. In these embodiments, if the preferred type of processing node is unavailable, the task can be assigned to a processing node of one of the back up choice types.

[0046] C. Other Resources

[0047] In some embodiments, a task definition may specify other types of required (or desired) resources. For example, a task definition may specify memory requirements, such as a minimum amount of memory, a maximum amount of memory, a type of memory, etc. Also for example, a task definition may specify input/output (I/O) requirements such as buffer requirements, I/O port requirements, etc. In some embodiments, I/O requirements can be specified, for example, in terms of a minimum buffer size, a maximum buffer size, a minimum throughput, a maximum throughput, a type of input, output, or I/O port, a specific input, output, or I/O port, etc. Examples of source code for specifying requirements of buffers will be described subsequently.

[0048] D. Port Aliases

[0049] In some embodiments, a task definition can include port aliases used for communicating between tasks, within a task, between a task and the operating system, etc. Ports can be, for example, I/O ports, registers, memories, sections of memories, etc., used for providing information to, or receiving information from, tasks. For instance, a task definition can include global alias names of ports for communicating with the task. Examples of source code for specifying port aliases will be described subsequently.

[0050] E. Task Loading

[0051] In some embodiments, a task definition can include requirements for loading the task. For example, a task definition can specify whether the task should be loaded for execution on a particular node, or within a particular group of nodes. Additionally, in some embodiments, a task definition can specify whether a task should be loaded for execution on a node on which another particular task or tasks is loaded for execution, or near a node or nodes on which another particular task or tasks is loaded for execution.

[0052] As described above, in some embodiments a task may comprise one or more modules. In these embodiments, a task definition can specify requirements relating to the node or nodes on which the modules should be loaded for execution. For instance, a task definition can specify that a particular module be loaded on a particular node or within a group of nodes. Also, a task definition can specify that two or more particular modules be loaded on a same node, or within one group of nodes. Similarly, a task definition can specify that two or more particular modules be loaded on different nodes, or on different groups of nodes. In some embodiments, a task definition can specify loading requirements for some modules while not specifying such requirements for other modules.

Examples

[0053] Examples of source code for implementing task definitions will now be described. It is to be understood that these examples are merely illustrative and are not limiting.

[0054] A. Module Definition

[0055] In some embodiments in which a task can comprise one or more modules, a task definition includes a module definition section. An example of source code included within a module definition section is provided below. In this example, a module definition section begins with a name indicating the module to be defined, followed by parentheses and brackets:

```
modSample("modSample_EntryPoint",
"m_node_files/modSample.mlf") {
    .
    .
    .
}
```

[0056] In the above example, a filename of an executable image of the module and an entry point within the file are specified within the parentheses. In particular, the filename of an executable images is "modSample.mlf" located in the directory "m_node_files." Additionally, the entry point is "modSample_EntryPoint." Within the brackets, other requirements of the module can be specified.required, as will be described subsequently.

[0057] In some embodiments, multiple file names can be specified, corresponding to alternative executable images of the module. For example, if different types of nodes require different formats, different code, etc., then one or more alternative executable images can be specified in case a particular node type is unavailable, for example, because the device on which the module is to be executed does not include it, because all nodes of this type have already been reserved by other modules, etc. In the following example, it is assumed that a device on which the module could be loaded might include two types of processing nodes: m-type nodes and a-type nodes. A required (or desired) type of node for a particular node can be specified by the filename extension of the executable image of the module. In this particular example, a ".mlf" extension indicates, for example, to an operating system, to a scheduler, etc., that the module should be loaded to an m-type node, whereas a ".alf" extension indicates that the module should be loaded to an a-type node:

```
modSample("modSample_EntryPoint", "m_node_files/modSample.mlf",
    "a_node_files/modSample.alf") {
    .
    .
    .
}
```

[0058] In the above example, the task definition specifies that the operating system, scheduler, etc., should first attempt to load the file named "modSample.mlf" to an m-type node (or, alternatively, to a node that supports an m-type format). If there is no such processing node available, then it should be attempted to load the file named "modSample.alf" to an appropriate processing node.

[0059] In some embodiments, multiple instances of a module can be loaded and executed.

[0060] An example of source code included within a module definition section that defines multiple images of a module is provided below. In this example, two instances are defined which reference the same executable image "modSample.mlf":

```
modSample1("modSample__EntryPoint",
"m__node__files/modSample.mlf") {
    .
    .
    .
}
modSample2("modSample__EntryPoint",
"m__node__files/modSample.mlf") {
    .
    .
    .
}
```

[0061] B. Module Resources Definition

[0062] The source code examples below illustrate one specific embodiment of a task definition that specifies node resources for a module. A keyword "cpu" followed by a number between 0 and 100 (inclusive) is used to specify a percentage of required processing resources of a node. In the following example, required processing node resources for the module whose executable image is included in the file "modSample.mlf" are specified. In particular, this module requires 25% of the processing node's processing power:

```
modSample("modSample__EntryPoint",
"m__node__files/modSample.mlf") {
    .
    .
    .
    resource( ) {
        .
        .
        .
        cpu 25
        .
        .
        .
    }
    .
    .
    .
}
```

[0063] The granularity of the number specifying processing power can vary with different implementations, different devices on which the module is to be executed, different types of processing nodes, etc. For example, the granularity can be in units of 1, 5, 10, 25, etc. If desired, a smaller granularity can also be used.

[0064] If multiple executable images of a module are defined, required processing resources can be specified as the same for both images, or specified individually. In the following example, required processing resources for two executable images of a module ("m_node_files/modSample.mlf" and "a_node_files/modSample.alf") are specified as having the same resource requirement:

```
modSample("modSample__EntryPoint", "m__node__files/modSample.mlf",
    "a__node__files/modSample.alf") {
    .
    .
    .
    resource( ) {
        .
        .
        .
        cpu 25
        .
        .
        .
    }
    .
    .
    .
}
```

[0065] In the following example, required processing resources for two executable images of a module ("m_node_files/modSample.mlf" and "anode_files/modSample.alf") are specified as having the different resource requirement:

```
modSample("modSample__EntryPoint", "m__node__files/modSample.mlf",
    "a__node__files/modSample.alf") {
    .
    .
    .
    resource("m__node__files/modSample.mlf") {
        .
        .
        .
        cpu 75
        .
        .
        .
    }
    resource("a__node__files/modSample.alf") {
        .
        .
        .
        cpu 25
        .
        .
        .
    }
    .
    .
    .
}
```

[0066] C. Module Overlay Definition

[0067] In some embodiments, use of overlays is permitted. A source code example of a task definition that specifies overlay requirements is provided below. In this example, the keyword "overlays" is used followed by parentheses and brackets. The parentheses can be used to specify a particular executable image of a module. If only one executable image has been defined, or if the overlays are the same for the different executable images, the parentheses can be left empty. Within the brackets, the entry points and files of one or more overlays can be specified:

```
modSample("modSample_EntryPoint",
"m_node_files/modSample.mlf") {
    .
    .
    .
    overlays( ) {
        "Overlay_EntryPoint", "m_node_files/modOverlay1.mlf"
        "Overlay_EntryPoint", "m_node_files/modOverlay2.mlf"
        "Overlay_EntryPoint", "m_node_files/modOverlay3.mlf"
    }
    .
    .
    .
}
```

[0068] In the above example, three overlays are defined. This information can be used, for example, to ensure that enough memory is reserved for the module when it is loaded. For instance, in the above example, an operating system could reserve an amount of memory greater than or equal to the size of the executable image "modSample.mlf" plus the size of the largest of the three overlay files.

[0069] D. Port Aliases

[0070] In some embodiments, a task definition includes a port alias section. An example of source code that illustrates port aliases is provided below. In this example, a alias section is followed by a module definition section:

```
inBuf_public 0
outBuf_public 0
.
.
.
modSample("modSample_EntryPoint",
"m_node_files/modSample.mlf") {
    inBuf_private inBuf_public
    outBuf_private outBuf_public
    .
    .
    .
}
```

[0071] In the above example, two global aliases are defined: "inBuf_public" and "outBu_fpublic." These aliases identify ports that can be used to communicate with the module "modSample." The "0" following each of these port aliases specify that the ports requirements are default values. Source code examples of specifying requirments of ports will be described subsequently.

[0072] Within the module definition section, the global alias names are linked to internal port names of the module: "inBuf_private" and "outBuf_private." This can be useful, for example, when multiple instances of the same module are to be loaded. And example of using port aliases with multiple module instances is provided below:

```
inBuf_public 0
xfrBuf_public 0
outBuf_public 0
.
.
.
modSample1("modSample_EntryPoint",
"m_node_files/modSample.mlf") {
    inBuf_private inBuf_public
    outBuf_private xfrBuf_public
    .
    .
    .
}
modSample2("modSample_EntryPoint",
"m_node_files/modSample.mlf") {
    inBuf_private xfrBuf_public
    outBuf_private outBuf_public
    .
    .
    .
}
```

[0073] In the above example, two instances of the same module are to be loaded, and three global aliases are defined. The private aliases of the two module instances are linked with the public aliases such that the ports will be interfaced as shown in FIG. **3**.

[0074] E. Port Resource Definition

[0075] As described above, in some embodiments, a task definition may specify input/output (I/O) requirements such as buffer requirements, I/O port requirements, etc. Examples of source code for specifying requirements of buffers are provided below. In the following example, requirements of two ports are specified:

```
inBuf_public 512
outBuf_public 256, 64, 4, 2046
.
.
.
modSample("modSample_EntryPoint",
"m_node_files/modSample.mlf") {
    inBuf_private inBuf_public
    outBuf_private outBuf_public
    .
    .
    .
}
```

[0076] In the above example, the port "inBuf_public" is specified to include 512 words of memory. The port "outBuf_public" is specified to include 256 words of memory, comprised of 4 separate buffers, each having 64 words. Additionally, the port "outBuf_public" is specified to be capable of handling a sustained data rate of at least 2046 kilobits of data per second. Any appropriate word size can be used depending upon the particular implementation (e.g., 8-bits, 16-bits, 32-bits, etc.). Additionally, port requirements need not be specified in terms of words. For example, port requirements could be specified in terms of bits, fixed-size blocks of words, etc. Similarly, a particular representation of a specified data rate is not required. In the following example, three ports are defined, each specifying an equivalent minimum data rate using different representations:

```
oneBuf_public 256, 64, 4, 2097152
twoBuf_public 256, 64, 4, 2048K
threeBuf_public 256, 64, 4, 2M
.
.
.
```

[0077] F. Module Loading

[0078] As described above, in some embodiments, a task definition can specify requirements for loading the task. A source code example is provided below that specifies loading requirements for a plurality of modules that comprise a task. In a specific embodiment, the tasks are to be loaded on a device that includes a group of processing nodes referred to as "ACM." Additionally, within the "ACM" processing nodes are organized into groups of four nodes, referred to as "quads." In the example below, loading requirements of 5 different modules ("modSample1," "modSample2," "modSample3," "modSample4," "modSample5") are specified:

```
ACM( ) {
    modSample1
    Quad( ) {
        modSample2
        Node( ) {
            modSample3
            modSample4
        }
        Node( ) {
            modSample5
        }
    }
}
```

[0079] In the above example, because modules "modSample1," "modSample2," "modSample3," "modSample4," and "modSample5" are included within the "ACM" brackets, these modules should be loaded onto the ACM group of processing nodes. Additionally, because module "modSample1" is not included within "Quad" or "Node" keyword brackets, then this module can be loaded on any "Quad" or "Node" in the "ACM" group, and without regard to any of the other modules.

[0080] Modules "modSample2," "modSample3," "modSample4," and "modSample5" are included within brackets of a "Quad" keyword. This specifies that "modSample2," "modSample3," "modSample4," and "modSample5" should be loaded on the same "Quad." Module "modSample2" is not included within "Node" keyword brackets. This specifies that this module can be loaded on any "Node" in the "Quad," and without regard to any of the other modules. Modules "modSample3" and "modSample4" are included within one set of "Node" keyword brackets, and module "modSample5" is included within another set of "Node" keyword brackets. This specifies that modules "modSample3" and "modSample4" should be loaded on the same processing node, and that module "modSample5" should be loaded on a different processing node than that of modules "modSample3" and "modSample4."

[0081] G. Example of a Task Definition

[0082] An example of a task definition is provided below. This example includes three sections: a "Port Aliases" section, a "Module Definition" section, and a "Module Loading" section:

```
//--------------------------------------------------------------------------------
// Section 1 - Port Aliases
inHostFifo      0x040
scatterBuf0     0x100
scatterBuf1     0x100
scatterBuf2     0x100
gatherBuf0      0x100
gatherBuf1      0x100
gatherBuf2      0x100
outHostFifo     0x400
tag0            0
tag1            0
tag2            0
ctlBuf          0
//--------------------------------------------------------------------------------
// Section 2 - Module Definitions
modInput("modInput_EntryPoint", "modGenData/modGenData.alf") {
    inBuf       inHostFifo
    outBuf      genData
    ctlBuf      SelectWhich
    resource("modGenData/modGenData.alf") {
        cpu     25
    }
}
modScatter("modScatter_EntryPoint", "modScatter/modScatter.alf",
                                    , "modScatter/modScatter.mlf") {
    inBuf       outBuf
    outBuf0     scatterBuf0
    outBuf1     scatterBuf1
    outBuf2     scatterBuf2
    resource("modScatter/modScatter.alf") {
        cpu     25
    }
    resource("modScatter/modScatter.mlf") {
        cpu     100
    }
}
modDecompress1("modDecompress_EntryPoint",
                "modDecompress/modDecompress.mlf",
                "modDecompress/modDecompress.alf") {
    inBuf       scatterBuf0
    outBuf      gatherBuf0
    tag         tag0
    resource("modDecompress/modDecompress.mlf") {
        cpu     25
    }
    resource("modDecompress/modDecompress.alf") {
        cpu     100
    }
}
modDecompress2("modDecompress_EntryPoint",
                "modDecompress/modDecompress.mlf") {
    inBuf       scatterBuf1
    outBuf      gatherBuf1
    tag         tag1
    resource( ) {
        cpu     25
    }
}
modDecompress3("modDecompress_EntryPoint",
                "modDecompress/modDecompress.mlf") {
    inBuf       scatterBuf2
    outBuf      gatherBuf2
    tag         tag2
    resource( ) {
        cpu     25
    }
}
modGather("modGather_EntryPoint", "modGather/modScatter.alf") {
    inBuf0      gatherBuf0
    inBuf1      gatherBuf1
    inBuf2      gatherBuf2
    outBuf      outHostFifo
    resource( ) {
        cpu     25
    }
    overlays( ) {
        "EntryPoint", "modGather/overlay1.alf"
        "EntryPoint", "modGather/overlay2.alf"
    }
}
//--------------------------------------------------------------------------------
// Section 3 - Module Loading
ACM( ) {
    modInput
```

-continued

```
    Quad( ) {
        modGather
        modDecompress3
        Node( ) {
            modScatter
        }
        Node( ) {
            modDecompress1
        }
        Node( ) {
            modDecompress2
        }
    }
}
```

[0083] While the above is a full description of the specific embodiments, various modifications, alternative constructions, and equivalents may be used. Therefore, the above description and illustrations should not be taken as limiting the scope of the present invention which is defined by the appended claims.

What is claimed is:

1. A computer program product comprising

a computer readable storage structure embodying computer readable code therein, the computer readable code comprising:

a task definition code that specifies requirements of a task adapted to be executed on a configurable device having a plurality of processing nodes, the task definition code including:

code that indicates processing node resources required by the task.

2. The computer program product of claim **1**, wherein the code that indicates processing node resources required by the task includes code that indicates processing power required of at least one processing node.

3. The computer program product of claim **1**, wherein the code that indicates processing node resources required by the task includes code that indicates a required execution time of at least a portion of the task.

4. The computer program product of claim **1**, wherein the code that indicates processing node resources required by the task includes code that indicates a maximum allowable latency of at least a portion of the task.

5. The computer program product of claim **1**, wherein the code that indicates processing node resources required by the task includes code that indicates a frequency of execution of at least a portion of the task.

6. The computer program product of claim **1**, wherein the task comprises one or more modules, and wherein code that indicates processing node resources required by the task includes code that indicates processing node resources required by at least one of the modules.

7. The computer program product of claim **1**, wherein the task definition code further includes code that indicates one or more types of processing nodes required to execute at least a portion of the task.

8. The computer program product of claim **1**, wherein the task definition code further includes code that indicates one or more processing nodes required to execute at least a portion of the task.

9. The computer program product of claim **1**, wherein the task definition code further includes code that indicates memory requirements of at least a portion of the task.

10. The computer program product of claim **1**, wherein the task definition code further includes code that indicates port requirements of at least a portion of the task.

11. The computer program product of claim **1**, wherein the task definition code further includes code that indicates port aliases.

12. The computer program product of claim **1**, wherein the task definition code further includes code that indicates requirements for loading of at least a portion of the task onto one or more processing nodes.

13. The computer program product of claim **12**, wherein the task comprises a plurality of modules, and wherein the code that indicates requirements for loading of at least a portion of the task includes code that specifies a group of processing nodes on which to load one or more modules.

14. The computer program product of claim **12**, wherein the task comprises a plurality of modules, and wherein the code that indicates requirements for loading of at least a portion of the task includes code that specifies at least two modules should be loaded on a single processing node.

15. The computer program product of claim **1**, wherein the task definition code further includes code that indicates the task to which the task definition is associated.

16. A computer data signal embodied in a carrier wave, the computer data signal comprising:

a task definition code that specifies requirements of a task adapted to be executed on a configurable device having a plurality of processing nodes, the task definition code including:

code that indicates processing node resources required by the task.

17. The computer data signal of claim **16**, wherein the task definition code further includes code that indicates one or more types of processing nodes required to execute at least a portion of the task.

18. The computer data signal of claim **16**, wherein the task definition code further includes code that indicates one or more processing nodes required to execute at least a portion of the task.

19. The computer data signal of claim **16**, wherein the task definition code further includes code that indicates memory requirements of at least a portion of the task.

20. The computer data signal of claim **16**, wherein the task definition code further includes code that indicates port requirements of at least a portion of the task.

21. The computer data signal of claim **16**, wherein the task definition code further includes code that indicates port aliases.

22. The computer data signal claim **16**, wherein the task definition code further includes code that indicates requirements for loading of at least a portion of the task onto one or more processing nodes.

23. A configurable device comprising:

a plurality of processing nodes;

a scheduler, coupled to the plurality of processing nodes, that assigns tasks to the processing nodes for execution; and

a memory, coupled to the scheduler, the memory including a task definition code that specifies requirements of at least one task adapted to be executed by the configurable device, the task definition code having code that indicates processing node resources required by the task.

* * * * *