Methods and apparatuses for bounding the processing domain in a symmetric multiprocessing system are provided. In various implementations, a particular computational task is "affined" to a particular processing unit. Subsequently, when the particular task is executed, the symmetric multiprocessing operating system ensures that the affined processing unit processes the instruction. When the affined processing unit is not processing the particular computational task, the symmetric multiprocessing operating system may cause the processing unit to process alternate instructions. With some implementations, a particular computational task is "linked" to a particular processing unit. Subsequently, when the particular task is executed, the symmetric multiprocessing operating system ensures that the bound processing unit processes the instruction. When the bound processing unit is not processing the particular computational instruction, the bound processing unit may enter a low power or idle state.
FIGURE 2
O403 Initialize The Processing Units.

O405 Boot The SMP OS.

O407 Load The Scheduler.

O409 Generate A Bound Computational Domain Queue.

O411 Move Selected Tasks To The BCD Queue.

O413 Create A Processing Domain Boundary For The BCD Queue.

**FIGURE 4**
DOMAIN BOUNDING FOR SYMMETRIC MULTIPROCESSING SYSTEMS

RELATED APPLICATIONS


FIELD OF THE INVENTION

[0002] The invention relates to the field of computing on multi-processor computer architectures. More particularly, various implementations of the invention are applicable to bounding the processing domains for a multi-processor embedded system.

BACKGROUND OF THE INVENTION

[0003] An embedded system may be described as a special purpose computing system designed to perform one or a few dedicated functions. Embedded systems are commonly used in consumer devices like personal digital assistants, mobile phones, videogame consoles, microwaves, washing machines, alarm systems, and digital cameras. In addition to the consumer space, embedded systems are used in nearly every industry, from telecommunications to manufacturing, and from transportation to medical devices. In fact, embedded systems are so commonly in use today that it is not feasible to exhaustively list specific examples.

[0004] The term “embedded system” does not have a precise definition, and determining what is and is not an embedded system can be difficult. For example, a general purpose computer, such as a laptop, is not typically characterized as an embedded system. However, a laptop is usually composed of a multitude of subsystems such as the hard disk drive, the motherboard, the optical drive, the video processing unit, and various communication devices. Many of the individual subsystems comprising the laptop may themselves be embedded systems.

[0005] The complexity of embedded systems can vary from, for example, systems with a single microcontroller chip and a light emitting diode to systems with multiple microprocessors and various peripheral communication interfaces and mechanical parts. Manufacturers of modern microprocessors are increasingly adding components and peripheral modules to their microprocessors, creating what may be thought of as embedded processors. This type of embedded system is often referred to as a system on a chip (SoC). A simple example of a system on chip is an application-specific integrated circuit (ASIC) packaged with a universal serial bus (USB) port. Additionally, embedded systems range from those having no user interface at all to those with full user interfaces similar to a desktop operating system.

[0006] There are many advantages to using embedded systems. For example, an embedded system typically is designed to do some specific task, as opposed to being a general purpose computer with a wide range of features for performing many different tasks. As a result, design engineers can optimize the embedded system for the desired task, which assists in reducing the size and cost of the device as well as increasing its reliability and performance.

[0007] Symmetric Multiprocessing

[0008] As stated above, embedded systems may often contain more than one processing unit. Embedded systems having more than one processing unit are often referred to as a multiprocessor system. In general, a multiprocessor computer system is any computing configuration that utilizes more than one processing unit. The processing units will typically share a memory. Additionally, one operating system is often used to control the entire system. In this type of arrangement, multiple computational tasks, or “instructions,” may be processed at the same time, such as, for example, one by each processing unit. This type of computing arrangement (i.e. where multiple processing units share a memory and are controlled by a single instance of an operating system) is often referred to as “symmetric multiprocessing” or SMP.

[0009] As indicated, an operating system is used to control the symmetric multiprocessing system. Controlling which processing units perform which tasks and when, is managed by the operating system, which typically operates on one of the processing units in the system. This operating system is often referred to as an SMP operating system or a symmetric multiprocessing operating system. As those of skill in the art can appreciate, various symmetric multiprocessing operating systems currently exist. For example, OS X, Linux, and various Unix-based operating systems are all capable of operating in a symmetric multiprocessing environment. Typically, a symmetric multiprocessing operating system allows a processor to work on any task, no matter the type of task or where the data for that task is located. Additionally, many symmetric multiprocessing operating systems move tasks between processors to balance the workload efficiently.

[0010] This type of task balancing and workload sharing may however, in some cases, be disadvantageous. This is particularly true in an embedded system where hardware and power constraints may dictate that particular processing units be employed to perform a particular type of task or operate on data located in a specific location.

SUMMARY OF THE INVENTION

[0011] Various implementations of the present invention provide methods and apparatuses for bounding the processing domain in a symmetric multiprocessing system. In various implementations, a particular computational task is “affined” to a particular processing unit. Subsequently, when the particular task is executed, the symmetric multiprocessing operating system ensures that the affined processing unit processes the instruction. When the affined processing unit is not processing the particular computational task, the symmetric multiprocessing operating system may cause the processing unit to process alternate instructions. With some implementations, a particular computational task is “linked” to a particular processing unit. Subsequently, when the particular task is executed, the symmetric multiprocessing operating system ensures that the bound processing unit processes the instruction. When the bound processing unit is not processing the particular computational instruction, the bound processing unit may enter a low power or idle state.

BRIEF DESCRIPTION OF THE DRAWINGS

[0012] The present invention will be described by way of illustrative embodiments shown in the accompanying drawings in which like references denote similar elements, and in which:
FIG. 1 shows an illustrative computing environment;
FIG. 2 shows a portion of the illustrative computing environment of FIG. 1 in greater detail;
FIG. 3 illustrates a conventional symmetric multiprocessor system;
FIG. 4 illustrates a method of bounding the processing domain of a symmetric multiprocessor system;
FIG. 5 illustrates a symmetric multiprocessor system according to various implementations of the present invention;
FIG. 6 illustrates the symmetric multiprocessing system of FIG. 5 in alternate detail; and
FIG. 7 illustrates the symmetric multiprocessing system of FIG. 5 in alternate detail.

DETAILED DESCRIPTION OF ILLUSTRATIVE IMPLEMENTATIONS

The operations of the disclosed implementations may be described herein in a particular sequential order. However, it should be understood that this manner of description encompasses rearrangements, unless a particular ordering is required by specific language set forth below. For example, operations described sequentially may in some cases be rearranged or performed concurrently. Moreover, for the sake of simplicity, the illustrated flowcharts and block diagrams typically do not show the various ways in which particular methods can be used in conjunction with other methods.

It should also be noted that the detailed description sometimes uses terms like “determine” to describe the disclosed methods. Such terms are often high-level abstractions of the actual operations that are performed. The actual operations that correspond to these terms will often vary depending on the particular implementation, and will be readily discernible by one of ordinary skill in the art.

The methods described herein can be implemented by software stored on a computer readable storage medium and executed on a computer. Furthermore, the selected methods could be executed on a single computer or a computer networked with another computer or computers. For clarity, only those aspects of the software germane to these disclosed methods are described; product details well known in the art are omitted.

As the techniques of the present invention may be implemented using software instructions executed by one or more programmable computing devices, the components and operation of a generic programmable computer system on which various implementations of the invention may be employed will first be described. Further, because of the complexity of some electronic design automation processes and the large size of many circuit designs, various electronic design automation tools are configured to operate on a computing system capable of simultaneously running multiple processing threads. The components and operation of a computer network having a host or master computer and one or more remote or slave computers therefore will be described with reference to FIG. 1. This operating environment is only one example of a suitable operating environment, however, and is not intended to suggest any limitation as to the scope of use or functionality of the invention.

In FIG. 1, the computer network 101 includes a master computer 103. In the illustrated example, the master computer 103 is a multi-processor computer that includes a plurality of input and output devices 105 and a memory 107. The input and output devices 105 may include any device for receiving input data from or providing output data to a user. The input devices may include, for example, a keyboard, microphone, scanner or pointing device for receiving input from a user. The output devices may then include a display monitor, speaker, printer or tactile feedback device. These devices and their connections are well known in the art, and thus will not be discussed at length here.

The memory 107 may similarly be implemented using any combination of computer readable media that can be accessed by the master computer 103. The computer readable media may include, for example, microcircuit memory devices such as read-write memory (RAM), read-only memory (ROM), electronically erasable and programmable read-only memory (EEPROM) or flash memory microcircuit devices, CD-ROM disks, digital video disks (DVD), or other optical storage devices. The computer readable media may also include magnetic cassettes, magnetic tapes, magnetic disks or other magnetic storage devices, punched media, holographic storage devices, or any other medium that can be used to store desired information.

As will be discussed in detail below, the master computer 103 runs a software application for performing one or more operations according to various examples of the invention. Accordingly, the memory 107 stores software instructions 109A that, when executed, will implement a software application for performing one or more operations. The memory 107 also stores data 109B to be used with the software application. In the illustrated embodiment, the data 109B contains process data that the software application uses to perform the operations, at least some of which may be parallel.

The master computer 103 also includes a plurality of processor units 111 and an interface device 113. The processor units 111 may be any type of processor device that can be programmed to execute the software instructions 109A, but will conventionally be a microprocessor device. For example, one or more of the processor units 111 may be a commercially generic programmable microprocessor, such as Intel® Pentium® or Xeon™ microprocessors, Advanced Micro Devices Athlon™ microprocessors or Motorola 68K/Coldfire® microprocessors. Alternately or additionally, one or more of the processor units 111 may be a custom-manufactured processor, such as a microprocessor designed to optimally perform specific types of mathematical operations. The interface device 113, the processor units 111, the memory 107 and the input/output devices 105 are connected together by a bus 115.

With some implementations of the invention, the master computing device 103 may employ one or more processing units 111 having more than one processor core. Accordingly, FIG. 2 illustrates an example of a multi-core processor unit 111 that may be employed with various embodiments of the invention. As seen in this figure, the processor unit 111 includes a plurality of processor cores 201. Each processor core 201 includes a computing engine 203 and a memory cache 205. As known to those of ordinary skill in the art, a computing engine contains logic devices for performing various computing functions, such as fetching software instructions and then performing the actions specified in the fetched instructions. These actions may include, for
example, adding, subtracting, multiplying, and comparing numbers, performing logical operations such as AND, OR, NOR and XOR, and retrieving data. Each computing engine may then use its corresponding memory cache to quickly store and retrieve data and/or instructions for execution.

Each processor core is connected to an interconnect. The particular construction of the interconnect may vary depending upon the architecture of the processor unit. With some processor cores, such as the Cell microprocessor created by Sony Corporation, Toshiba Corporation and IBM Corporation, the interconnect may be implemented as an interconnect bus. With other processor units, however, such as the Opteron™ and Athlon™ dual-core processors available from Advanced Micro Devices of Sunnyvale, Calif., the interconnect may be implemented as a system request interface device. In any case, the processor cores communicate through the interconnect with an input/output interfaces and a memory controller. The input/output interface provides a communication interface between the processor unit and the bus. Similarly, the memory controller controls the exchange of information between the processor unit and the system memory. With some implementations of the invention, the processor units may include additional components, such as a high-level cache memory accessible shared by the processor cores.

While FIG. 2 shows one illustration of a processor unit that may be employed by some embodiments of the invention, it should be appreciated that this illustration is representative only, and is not intended to be limiting. For example, some embodiments of the invention may employ a master computer with one or more cell processors. The Cell processor employs multiple input/output interfaces and multiple memory controllers. Also, the Cell processor has nine different processor cores of different types. More particularly, it has six or more synergistic processor elements (SPEs) and a power processor element (PPE). Each synergistic processor element has a vector-type computing engine with 128x128 bit registers, four single-precision floating point computational units, four integer computational units, and a 256KB local store memory that stores both instructions and data. The power processor element then controls that tasks performed by the synergistic processor elements. Because of its configuration, the Cell processor can perform some mathematical operations, such as the calculation of fast Fourier transforms (FFTs), at substantially higher speeds than many conventional processors.

It also should be appreciated that, with some implementations, a multi-core processor unit can be used in lieu of multiple, separate processor units. For example, rather than employing six separate processor units, an alternate implementation of the invention may employ a single processor unit having six cores, two multi-core processor units each having three cores, a multi-core processor unit with four cores together with two separate single-core processor units, etc.

Returning now to FIG. 1, the interface device allows the master computer to communicate with the slave computers through a communication interface. The communication interface may be any suitable type of interface including, for example, a conventional wired network connection or an optically transmissive wired network connection. The communication interface may also be a wireless connection, such as a wireless optical connection, a radio frequency connection, an infrared connection, or even an acoustic connection. The interface device translates data and control signals from the master computer and each of the slave computers into network messages according to one or more communication protocols, such as the transmission control protocol (TCP), the user datagram protocol (UDP), and the Internet protocol (IP). These and other conventional communication protocols are well known in the art, and thus will not be discussed here in more detail.

Each slave computer may include a memory, a processor unit, an interface device, and, optionally, one more input/output devices connected together by a system bus. As with the master computer, the optional input/output devices may be any conventional or custom-manufactured programmable processor device. For example, one or more of the processor units may be commercially generic programmable microprocessors, such as Intel® Pentium® or Xeon™ microprocessors, Advanced Micro Devices Athlon™ microprocessors or Motorola 68K/Coldfire® microprocessors. Alternately, one or more of the processor units may be custom-manufactured processors, such as microprocessors designed to optimally perform specific types of mathematical operations. Still further, one or more of the processor units may have more than one core, as described with reference to FIG. 2 above. For example, with some implementations of the invention, one or more of the processor units may be a Cell processor. The memory may be implemented using any combination of the computer readable media discussed above. Like the interface device, the interface devices allow the slave computers to communicate with the master computer over the communication interface.

In the illustrated example, the master computer is a multi-processor unit computer with multiple processor units, while each slave computer has a single processor unit. It should be noted, however, that alternate implementations of the invention may employ a master computer having single processor unit. Further, one or more of the slave computers may have multiple processor units, depending upon their intended use, as previously discussed. Also, while only a single interface device is illustrated for both the master computer and the slave computers, it should be noted that, with alternate embodiments of the invention, either the computer, one or more of the slave computers, or some combination of both may use two or more different interface devices for communicating over multiple communication interfaces.

Furthermore, it is to be appreciated that, although in the example, the master computer and the slave computers are shown as individual discrete units, some implementations may package the master computer and the slave computers into a single unit, such as, for example, a System-on-Chip device.

With various examples of the invention, the master computer may be connected to one or more external data storage devices. These external data storage devices may be implemented using any combination of computer readable media that can be accessed by the master computer. The
computer readable media may include, for example, microcircuit memory devices such as read-write memory (RAM), read-only memory (ROM), electronically erasable and programmable read-only memory (EEPROM) or flash memory microcircuit devices, CD-ROM disks, digital video disks (DVD), or other optical storage devices. The computer readable media may also include magnetic cassettes, magnetic tapes, magnetic disks or other magnetic storage devices, punched media, holographic storage devices, or any other medium that can be used to store desired information. According to some implementations of the invention, one or more of the slave computers may alternate or additions be connected to one or more external data storage devices. Typically, these external data storage devices will include data storage devices that also are connected to the master computer, but they also may be different from any data storage devices accessible by the master computer.

It should also be appreciated that the description of the computer network illustrated in FIG. 1 and FIG. 2 is provided as an example only, and it is not intended to suggest any limitation as to the scope of use or functionality of alternate embodiments of the invention.

Symmetric Multiprocessing Systems

As indicated above, a conventional symmetric multiprocessing system includes a plurality of processing units, capable of independently executing various tasks. For example, FIG. 3 illustrates a conventional symmetric multiprocessing system. The system includes a computing environment, including processing units. In various implementations of the invention, the computing environment may be formed by the computer network of FIG. 1. As such, the processing units would comprise the processor units and shown in the figure. With some implementations, the computing environment may be formed by the master computer. Accordingly, the processing units would comprise the processor units. As can be appreciated, various components of the computing environment are not shown in this example. For example, the computing environment would include a memory component, which is in some cases, may be implemented by the memory shown in FIG. 1.

The system also includes a symmetric multiprocessing scheduler having a symmetric multiprocessing queue. Furthermore, as can be seen, the symmetric multiprocessing queue includes tasks. As detailed above, in a conventional symmetric multiprocessing system, any of the tasks may be executed on any of the processing units. The symmetric multiprocessing scheduler may assign particular tasks to any of the processing units, and may change or move the assignments dynamically to balance the computational load efficiently.

However, as indicated above, this has some disadvantages. One such disadvantage is that processing units are increasingly more specific to particular tasks. This is often the case in embedded systems, where a particular processing unit may have been designed for a specific function, such as, for example, video encoding. Additionally, ones of the processing units may have much higher power consumption needs than other ones of the processing units. As such, use of these processing units could be better controlled to manage power consumption for the system.

As used herein, a processing unit may be either a microprocessor or a core within a multi-core microprocessor, such as, for example, the processor unit and the processor core respectively. Furthermore, applicants would like to point out that although in practice, a distinction between a symmetric computing architecture (i.e., homogeneous processing units that share memory) and an asymmetric computing architecture (i.e., heterogeneous processing units that share memory) may be made; herein, when referencing a symmetric multiprocessing system, not all processing units must be homogeneous. For example, as used herein, a symmetric multiprocessing system may have a combination of single core microprocessors and multi-core microprocessors. Furthermore, the microprocessors may have different hardware specifications. Still, further, the microprocessors may have different computer processor architectures.

FIG. 4 illustrates a method for bounding the processing domain of a symmetric multiprocessing system. For example, the method may be implemented in conjunction with the example symmetric multiprocessing system shown in FIG. 5. As can be seen from these figures, the symmetric multiprocessing system includes, among other items, a computing environment having processing units. In various implementations, the symmetric multiprocessing system may be formed by modifying the symmetric multiprocessing system shown in FIG. 3. Still, in some implementations, the symmetric multiprocessing system may be formed by utilizing the computing network, or alternatively, from the master computer as the computing environment.

Returning to FIG. 4, the method includes an operation for initializing the processing units within the symmetric multiprocessing system and an operation for booting a symmetric multiprocessing operating system on one or more of the processing units. As shown in this Figure, the symmetric multiprocessing operating system is booted onto the processing unit. The processing unit loads the operating system (e.g., the processing unit in this example) is often referred to as the “boot processor.” In various implementations of the invention, the boot processor is used exclusively by the symmetric multiprocessing operating system for operations related to managing the symmetric multiprocessing system. In alternative implementations, the boot processor is used to load the operating system, but is not used exclusively for operations related to managing the symmetric multiprocessing system. Accordingly, in some implementations, the boot processor is available for general computing tasks un-related to operating system management. With some implementations of the invention, the operation initializes all the processing units. With alternative implementations, the operation initializes only the boot processor.

The method further includes an operation for loading the scheduler. As can be seen from FIG. 5, the scheduler includes a symmetric multiprocessing queue. As can be further seen from this figure, the system additionally includes a user application having tasks. As used herein, the tasks may be explicit instructions that processing units may directly execute. Alternatively, the tasks may be higher level operations that the symmetric multiprocessing operating system will translate into instructions that the processor units may execute.
Although FIG. 5 illustrates a single user application 515, and one set of tasks 517, in various implementations, more than one user application 515 may be executed by the symmetric multiprocessing operating system 507. Additionally, in some implementations, the user application 515 may have multiple sets of tasks 517. Further still, as can be appreciated by those of skill in the art, the set of tasks 517 is typically not static. More particularly, the set of tasks 517 changes as the user application is executed.

The method 401 additionally includes an operation 409 for generating a bound computational domain queue 513 and an operation 411 for moving selected tasks 517 to the bound computational domain queue 513. In various implementations of the invention, as a user application 515 is loaded by the symmetric multiprocessing operating system 509 and tasks 517 associated with the user application 515 are identified, all the tasks 517 may be initially loaded into the symmetric multiprocessing queue 511. More particularly, when the scheduler 509 is first loaded by the operation 407, the scheduler may only include the symmetric multiprocessing queue 511, which will include all of the tasks 517.

In various implementations of the invention, the operation 409 and the operation 411 are performed as a result of some user input. With some implementations, the operation 409 and the operation 411 are triggered without user input, such as, for example, based upon the type of user application 515 or the type of task 517. In various implementations of the invention, the operations 409 and 411 may be repeated a number of times, resulting in more than one bound computational domain queue 513 being created within the scheduler 509.

The method 401 further includes an operation 413 for forming a processing domain boundary for the bound computational domain queue 513. As stated above, in various implementations, an “affinity” is created between a bound computational domain queue 513 and one or more processing units 505. Alternatively, a “link” is created between a bound computational domain queue 513 and one or more processing units 505. These example processing domain boundaries are discussed in greater detail below.

Bound Computational Domain with Affinity

In various implementations, the operation 413 “affines” one or more of the processing units 505 to the bound computational domain queue 513. Tasks 517 included in a bound computational domain queue 513 that is “affined” to a particular processing unit 505 are said to be affinity to that particular processing unit 505. Tasks 517 that are affinity to a particular processing unit 505 are given “priority” by the scheduler 509 to execute on that particular processing unit 505. However, when tasks 517 having an affinity for the selected processing unit 505 are not being executed, the processing unit 505 is available for scheduling non-affined tasks 517 by the scheduler 509. Priority of execution may be shown by the scheduler 509 by transferring execution of non-affined tasks 517 to idle processing units 505 when affinity tasks 517 need to be executed. Alternatively, priority may be shown by stalling execution of the affinity task 517 until the affinity processing unit 505 is available for executing tasks 517.

In some implementations, a single processing unit 505 is affinity to a bound computational domain queue 513 by the operations 413. With some implementations, multiple processing units 505 are affinity to a bound computational domain queue 513. For example, FIG. 6 illustrates the symmetric multiprocessing system 501 of FIG. 5, where the bound computational domain queue 513 has been affinity to the processing unit 505/ii and the processing unit 505/n, as illustrated by the boundary 503. As can be seen from this figure, the user application 515 is not shown. However, the tasks 517 from the user application 515 have been moved into the symmetric multiprocessing queue 511 and the bound computational domain queue 513.

As a result of the affinity created by the operation 415 (as illustrated by the boundary 603) the scheduler 509 may assign the tasks 517/iv, 517/v, and 517/vi to execute on either of the processing units 505/iii or 505/n. Additionally, the scheduler 509 may assign the tasks 517/ii or 517/iii to execute on the processing unit 505/i. Alternatively, if the processing unit 505/iii is not executing tasks 517 from the bound computational domain queue 513, tasks 517 from the symmetric multiprocessing queue 511 may be executed on the processing unit 505/iii. Alternatively still, if the processing unit 505/n is not executing tasks 517 from the bound computational domain queue 513, tasks 517 from the symmetric multiprocessing queue 511 may be executed on the processing unit 505/n.

Bound Computational Domain with Link

As stated above, with some implementations, the operation 413 may “link” one or more the processing units 505 to a bound computational domain queue 513. Processing units 505 that have been linked to a particular task 517 or set of tasks 517 can only execute those tasks 517. When there are no linked tasks to execute, the processor remains idle, as opposed to becoming available for scheduling as in the case of an affinity processing unit 505.

FIG. 7 illustrates the symmetric multiprocessing system 501 shown in FIG. 5 and FIG. 6. However, FIG. 7 includes a boundary 703 that shows a link, as opposed to an affinity as shown by the boundary 503 in FIG. 6. As can be seen from FIG. 7, boundary 603 isolates the processing units 505/ii or 505/n to the bound computational domain 513. As a result, only the tasks 517/iv, 517/v, and 517/vi may be executed by the processing units 505/iii and 505/n.

In various implementations, as opposed to bounding a queue of tasks 517, such as, for example, the bounded computational domain queue 513, as described above, the processing domain for individual tasks 517 may be bound. For example, the operation 415 may directly affinity the task 517/vi with the processing unit 505/iii. As opposed to including the task 517/vi into a bound computational domain queue 513 and then bounding the processing domain for the queue 513.

Conclusion

Although certain devices and methods have been described above in terms of the illustrative embodiments, the person of ordinary skill in the art will recognize that other embodiments, examples, substitutions, modifications and alterations are possible. It is intended that the following claims cover such other embodiments, examples, substitutions, modifications and alterations within the spirit and scope of the claims.

What is claimed is:

1. A computer-implemented method for bounding the processing domain in a symmetric multiprocessing system, the method comprising:

   a. identifying a symmetric multiprocessing system, the symmetric multiprocessing system including a plurality of processing units;
identifying a plurality of tasks to be scheduled for execution by the symmetric multiprocessing system;
forming a computationally bound task queue;
moving selected ones of the plurality of tasks to be scheduled to the computationally bound task queue; and
bounding the processing domain for the computationally bound task queue.
2. The computer-implemented method recited in claim 1, further comprising:
caus[ing] a symmetric multiprocessing operating system to boot onto a one of the plurality of processing units.
3. The computer-implemented method recited in claim 2, further comprising:
loading a symmetric multiprocessing operating system scheduler.
4. The computer-implemented method recited in claim 3, the method act for identifying a plurality of tasks to be scheduled for execution by the symmetric multiprocessing operating system comprising:
identifying a symmetric multiprocessing task queue within the symmetric multiprocessing operating system scheduler;
identifying a plurality of tasks within the symmetric multiprocessing task queue; and
designating the identified tasks as the plurality of tasks to be scheduled.
5. The computer-implemented method recited in claim 4, the method act of forming a computationally bound task queue comprising:
recieving an instruction from a user of the symmetric multiprocessing system to create a bound computational domain; and
forming the task queue within the symmetric multiprocessing operating system scheduler to represent the computationally bound task queue.
6. The computer-implemented method recited in claim 5, the instruction including a listing of one or more of the plurality of tasks to be scheduled and the method act of moving selected ones of the plurality of tasks to be scheduled to the computationally bound task queue comprising:
adding the one or more of the plurality of tasks to be scheduled listed in the instruction to the computationally bound task queue; and
removing the one or more of the plurality of tasks to be scheduled listed in the instruction from the symmetric multiprocessing task queue.
7. The computer-implemented method recited in claim 6, the instruction including a listing of one or more of the plurality processing units and the method act of bounding the processing domain for the computationally bound task queue comprising affining the computationally bound task queue to the one or more of the plurality of processing units listed in the instruction.
8. The computer-implemented method recited in claim 6, the instruction including a listing of one or more of the plurality processing units and the method act of bounding the processing domain for the computationally bound task queue comprising linking the computationally bound task queue to the one or more of the plurality of processing units listed in the instruction.
9. The computer-implemented method recited in claim 1, further comprising:
forming a second computationally bound task queue;
moving selected ones of the plurality of tasks to the second computationally bound task queue; and
bounding the processing domain for the computationally bound task queue.
10. The computer-implemented method recited in claim 1, further comprising:
unbounding the computationally bound task queue; and
removing the computationally bound task queue from the symmetric multiprocessing system.
11. One or more tangible computer-readable media, having computer executable instructions for bounding the processing domain in a symmetric multiprocessing system stored thereon, the computer executable instructions comprising:
caus[ing] a computer to perform a set of operations; and
wherein the set of operations include:
identifying a symmetric multiprocessing system, the symmetric multiprocessing system including a plurality of processing units;
identifying a plurality of tasks to be scheduled for execution by the symmetric multiprocessing system;
forming a computationally bound task queue;
moving selected ones of the plurality of tasks to be scheduled to the computationally bound task queue; and
bounding the processing domain for the computationally bound task queue.
12. The one or more tangible computer-readable media recited in claim 11, the symmetric multiprocessing system including a symmetric multiprocessing operating system scheduler and the operation for identifying a plurality of tasks to be scheduled for execution by the symmetric multiprocessing operating system comprising:
identifying a symmetric multiprocessing task queue;
identifying a plurality of tasks within the symmetric multiprocessing task queue; and
designating the identified tasks as the plurality of tasks to be scheduled.
13. The one or more tangible computer-readable media recited in claim 12, the operation for forming a computationally bound task queue comprising:
recieving an instruction from a user of the symmetric multiprocessing system to create a bound computational domain; and
forming the task queue within the symmetric multiprocessing operating system scheduler to represent the computationally bound task queue.
14. The one or more tangible computer-readable media recited in claim 13, the instruction including a listing of one or more of the plurality of tasks to be scheduled and the operation for moving selected ones of the plurality of tasks to be scheduled to the computationally bound task queue comprising:
adding the one or more of the plurality of tasks to be scheduled listed in the instruction to the computationally bound task queue; and
removing the one or more of the plurality of tasks to be scheduled listed in the instruction from the symmetric multiprocessing task queue.
15. The one or more tangible computer-readable media recited in claim 14, the instruction including a listing of one or more of the plurality processing units and the operation for bounding the processing domain for the computationally bound task queue comprising affining the computationally bound task queue.
bound task queue to the one or more of the plurality of processing units listed in the instruction.

16. The one or more tangible computer-readable media recited in claim 14, the instruction including a listing of one or more of the plurality processing units and the operation for bounding the processing domain for the computationally bound task queue comprising linking the computationally bound task queue to the one or more of the plurality of processing units listed in the instruction.

17. A symmetric multiprocessing system adapted to allowing bounded processing, the system comprising:
   a plurality of processing units;
   a plurality of tasks to be scheduled for execution by the system; and
   a memory including a set of instructions that cause the system to perform:
   forming a computationally bound task queue;
   moving selected ones of the plurality of tasks to be scheduled to the computationally bound task queue; and
   bounding the processing domain for the computationally bound task queue.

18. The symmetric multiprocessing system recited in claim 17, wherein the set of instructions are included in an application programming interface.

19. The symmetric multiprocessing system recited in claim 18, the instruction for bounding the processing domain for a computationally bound task queue comprising:
   receiving a listing of one or more of the plurality processing units
   receiving a boundary relationship affining the computationally bound task queue to the one or more of the plurality of processing units listed in the instruction.

20. The symmetric multiprocessing system recited in claim 18, the instruction for bounding the processing domain for a computationally bound task queue comprising:
   receiving a listing of one or more of the plurality processing units
   receiving a boundary relationship linking the computationally bound task queue to the one or more of the plurality of processing units listed in the instruction.