US007093147B2

(12) **United States Patent**
Farkas et al.

(10) **Patent No.: US 7,093,147 B2**
(45) **Date of Patent: Aug. 15, 2006**

(54) **DYNAMICALLY SELECTING PROCESSOR CORES FOR OVERALL POWER EFFICIENCY**

(75) Inventors: **Keith Farkas**, San Carlos, CA (US); **Norman P. Jouppi**, Palo Alto, CA (US); **Robert N. Mayo**, Mountain View, CA (US); **Parthasarathy Ranganathan**, Palo Alto, CA (US)

(73) Assignee: **Hewlett-Packard Development Company, L.P.**, Houston, TX (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 461 days.

(21) Appl. No.: **10/423,397**

(22) Filed: **Apr. 25, 2003**

(65) **Prior Publication Data**

US 2004/0215987 A1 Oct. 28, 2004

(51) **Int. Cl.**
G06F 1/32 (2006.01)

(52) **U.S. Cl.** **713/320**

(58) **Field of Classification Search** 713/320
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,913,068 A * 6/1999 Matoba 713/322
6,804,632 B1 * 10/2004 Orenstien et al. 702/188

OTHER PUBLICATIONS

R. Kumar et al., "Processor Power Reduction Via Single-ISA Heterogeneous Multi-core Architectures", In Computer Architecture Letters, vol. 2, Apr. 2003.*
R. Kumar et al., "A Multi-Core Approach to Addressing the Energy-Complexity Problem in Microprocessors", In Work on Complexity-Effective Design, Jun. 2003.*

* cited by examiner

Primary Examiner—Chun Cao
Assistant Examiner—Albert Wang

(57) **ABSTRACT**

A computer system for conserving operating power includes a number of computer hardware processor cores that differ amongst themselves in at least in their respective operating power requirements and processing capabilities. A monitor gathers performance metric information from each of the computer hardware processor cores that is specific to a particular run of application software then executing. A workload transfer mechanism transfers the executing application software to a second computer hardware processor core in a search for reduced operating power. A transfer delay mechanism is connected to delay a subsequent transfer of the executing application software if the system operating power may be conserved by such delay.

16 Claims, 2 Drawing Sheets

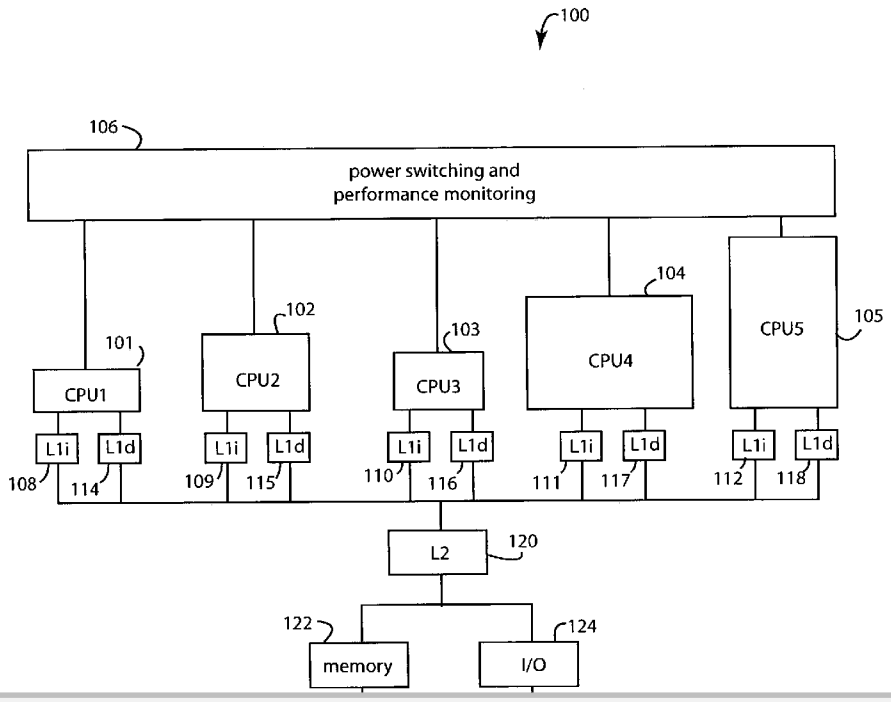


Fig. 1

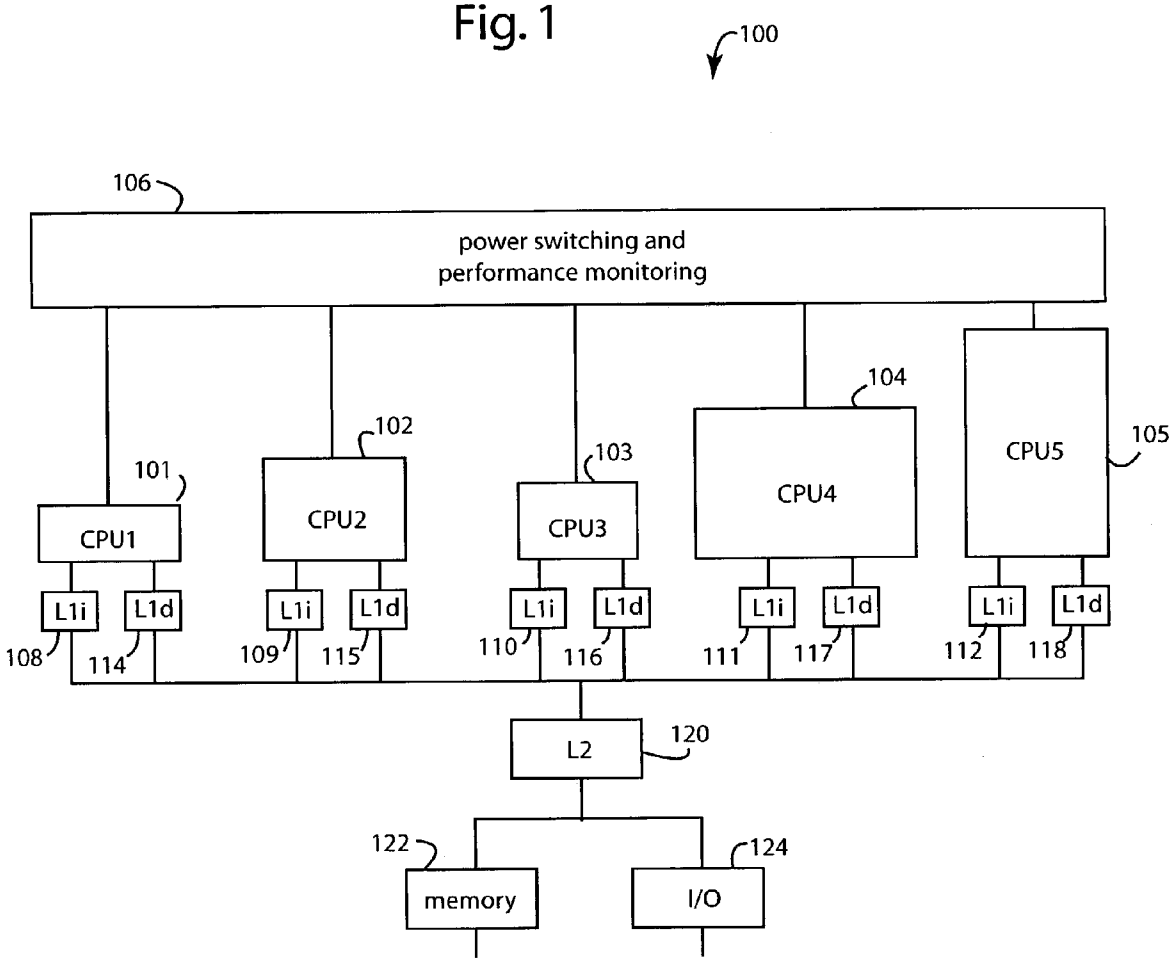


Fig. 2

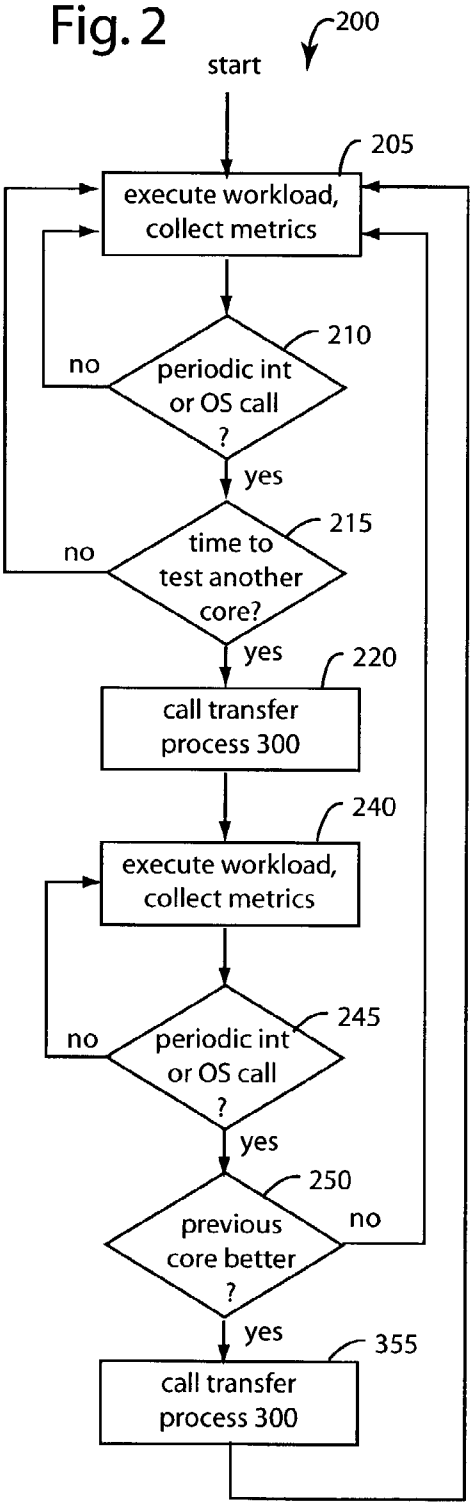
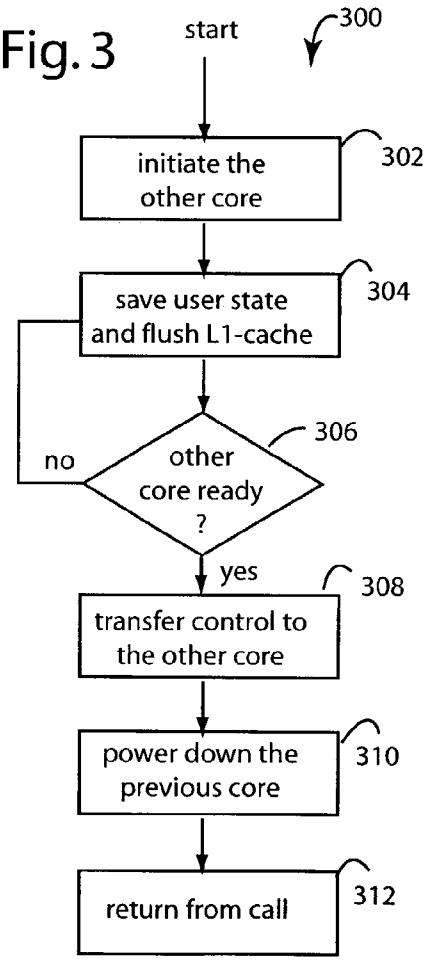


Fig. 3



DYNAMICALLY SELECTING PROCESSOR CORES FOR OVERALL POWER EFFICIENCY

FIELD OF THE INVENTION

The present invention relates to computer systems, and more specifically to methods and devices for reducing power use by dispatching processing jobs to the more energy-efficient processor core, in a pool of different-capability processor cores, that nevertheless provides acceptable performance.

BACKGROUND OF THE INVENTION

Computer software application programs do not always require the high-capability computing hardware resources that are at their disposal. But if some critical code passage or whole application program must run at maximum efficiency, conventional systems dedicate the necessary computing hardware full time. In a few prior art multiprocessor systems that run applications that can be split and paralleled, pools of identical processor cores can be added in sufficient numbers to get the job done.

Some waste can be involved in the mismatching of software with modest resource requirements on high performance hardware platforms. When there is only one processor core available for all processing jobs, the waste of computing resources and power to operate them is unavoidable. High performance hardware is usually associated with large demands on operating power input. If such high performance is going to waste much of the time, the marginal operating power needed over more modest equipment is pure cost with no benefit.

Since their introduction in the 1970's, microprocessors and microcomputer systems have been providing ever more increasing levels of performance, reliability, and capability. Every few years since then has seen the microprocessor evolve to new, higher levels. Clock speeds got higher, memory subsystems, cache memories, and peripherals were brought in on-chip as semiconductor technology advances permitted. Complex instruction set computers (CISC) and reduced instruction set computers (RISC) evolved, and instruction and data bus widths reached 32-bits, 64-bits, and even 128-bits.

Device technologies have been changing. The first Intel microprocessors, e.g., the 4004, used p-channel metal oxide semiconductor (PMOS) technology. Later processors used n-channel metal oxide semiconductor (NMOS) technology. An RCA microprocessor family, the 1802, used low-power complementary metal oxide semiconductor (CMOS) technology. Some very high performance microprocessors in the 1970's and later used bipolar transistor technology. Today's MOS technology used in microprocessors has high leakage currents that require the operating power to actually be interrupted, or switched off, in order to reduce power consumption completely in inactive circuits.

In general, higher clock speeds and denser functionality has meant increased power consumption and hence dissipation. Such power dissipation causes undesirable heating and, in battery-operated portable systems, leads to reduced battery life. Constantly using a processor that uses a lot of

SUMMARY OF THE INVENTION

An object of the present invention is to provide a method for reducing average power consumption in computing devices.

Another object of the present invention is to provide a computer system with reduced average power consumption.

Briefly, a computer system embodiment of the present invention comprises a number of processor cores consigned to a pool. Such processor cores differ in their respective levels and mix of power consumption, resources, performance, and other important measures. These processor cores can be arranged in a linear order according to estimates of one or more of these measures. An operating system associated with the processor core pool dispatches the execution of application programs to various processor cores and runs empirical tests. In general, the target processor core from the pool being sought for the job is the one that consumes a minimum of power and still yields acceptable performance. Such balance is determined statically for each workload based on data from prior executions of the workload. Alternatively, such balance is struck dynamically and empirically determined at run-time. Metrics are collected on how well an application runs on a particular processor core in the pool, for example during a one millisecond test period. If the current processor core is yielding better results than a previous processor core, then the job will not be transferred, and will be allowed to continue executing. If not, the job can be returned to the previous processor core in the ordered pool or a next processor core can be tried. The resource requirements between application programs can vary, as well as the requirements at different times within a single application.

An advantage of the present invention is that a system is provided that can conserve battery power in portable computers.

Another advantage of the present invention is that a method is provided for conserving operating power and reducing power supply demands.

These and other objects and advantages of the present invention will no doubt become obvious to those of ordinary skill in the art after having read the following detailed description of the preferred embodiment as illustrated in the drawing figures.

DESCRIPTION OF THE DRAWINGS

FIG. **1** is a functional block diagram of an embodiment of the present invention comprising multiple processor cores;

FIG. **2** is a flowchart diagram of a process embodiment of the present invention for transferring software jobs amongst dissimilar cores in a pool of multiple processor cores; and

FIG. **3** is a flowchart diagram of a subroutine that is called by the process of FIG. **2** and that transfers program control between processor cores.

DETAILED DESCRIPTION OF THE EMBODIMENTS

FIG. **1** illustrates a multi-core processor system embodiment of the present invention, and is referred to herein by the general reference numeral **100**. Multi-core processor system **100** is a heterogeneous multi-core and core-switching implementation in a chip-level multi-core processor (CMP) with multiple, diverse processor cores that all execute the same

performance and energy efficiency levels for the same application software. The operating system software tries to match the applications to the different cores during an application's execution to make the best use of the available hardware while maximizing energy efficiency at a given minimum performance level.

The system **100** hosts an operating system and application software that can execute single-threaded or multi-threaded. The operating system dispatches processing jobs to individual processor cores that differ in their power consumption, available resources, relative speeds, and other important measures. Such dissimilar processor cores are represented here in FIG. **1** as CPU1 **101**, CPU2 **102**, CPU3 **103**, CPU4 **104**, and CPU5 **105**. A minimum of two dissimilar processor cores can yield the benefits of the present invention if they differ in their respective power consumptions and one other critical measure, e.g., through-put. Therefore, showing the five CPU's **101**–**105** in FIG. **1** is merely for purposes of discussion here. Such processor cores can execute more than one process or thread at a time.

The multi-core processor system **100** comprises a pool of dissimilar processor cores **101**–**105** that receive their respective power supplies from a power switch and monitor **106**. Such also provides monitoring information reported as metrics by each of the processor cores **101**–**105** during their respective execution of software programs. The metrics can include number of cache misses, etc.

Each processor core **101**–**105** has a corresponding first level instruction cache (L1i) **108**–**112**, and a corresponding first level data cache (L1d) **114**–**118**. These all share a common second level cache (L2) **120**, a main memory **122**, and input/output (I/O) device **124**. Operating system and application software execute from main memory **120** and are cached up through to the respective second and first level caches to processor cores **101**–**105**.

A timer is used to periodically interrupt the operating system, e.g., every one hundred time intervals. This interrupt invokes a transfer to and a test of one of the different cores, ordered according to some metric, for its energy or energy-delay product when running the current application software. For example, the test can sample the execution of the application for 1–2 time intervals. If the test of such different core results in a better energy-delay product metric than the previous core yielded, then the application software continues to execute on the new core. Otherwise, the application software is migrated back to the previous core, where it continues its execution from the point it reached before being migrated back to the previous core.

In order to gauge the impact on the energy-delay product, data on energy consumption is needed for each of the processor cores **101**–**105**. A mechanism is needed to determine whether to migrate the program executing workloads between the processor cores **101**–**105**. The migration or transfer of the program executing workloads needs to be accomplished with a minimal impact on any other performance metrics of interest.

A mechanism identifies the energy consumed by the different cores as a function of the workloads running on them. The metrics of interest may either be the total energy consumed by the system, the energy-delay product of the system, the peak power of the system, etc. The decision to migrate the workloads can use the metrics determined by the energy data, as well as other additional user-defined or workload-defined metrics. Such migration can be static or hierarchy, or more complicated ways to ensure that any performance loss is minimized.

FIG. **2** represents a method embodiment of the present invention for selecting which core to run in multi-core system **100** in FIG. **1**. Such method is referred to herein by the general reference numeral **200**. The method **200** is preferably implemented as a part of an operating system for multi-core system **100**.

Method **200** begins with a step **205** that collects statistics or metrics as a workload executes on a given processor core. The statistics relate to its execution, power consumption, performance, and other metrics. A step **210** continues this monitoring process until a periodic interrupt occurs. Interrupts can be generated by a timer, an operating system (OS) call, etc. In a step **215**, such periodic interrupt is serviced, and check is made to see if it is time to evaluate how well the workload executes on another core. The other core will differ, e.g., in a greater number of hardware resources, or one that is more energy efficient.

If it is not time to try another core, then control returns to continue executing on the present core. If it is time to try another core, then control passes to a process **300** (FIG. **3**).

In a step **240**, as a workload executes on a given processor core, statistics are collected about its execution, power consumption, performance, and other metrics. A step **245** continues monitoring until a periodic interrupt occurs. A timer or an operating system (OS) call can be used to generate these interrupts. In a step **250**, the interrupt is serviced, and an analysis is made to determine if the performance with the previous core had been better. If not, and the current core is determined to be better performing, the workload continues executing where it was, e.g., in steps **205** and **210** until a next interrupt occurs.

If, however, the previous core was better performing according to the metrics, a step **255** calls to transfer the workload back to the original processor core, using process **300**. Once the transfer is completed, the workload returns to executing steps **205** and **210**, e.g., until a next interrupt is detected in step **215**.

Referring now to FIG. **3**, a transfer-workload-to-another-core process **300** begins with a step **302** in which the other core is powered up. In a step **304**, the state of the application is saved to memory, and the cache of the current processor core is flushed. In a step **306**, a test is made repeatedly in a loop to determine if the other core is ready to begin executing instructions. When it is ready, a step **308** transfers software control to the other core. The other core executes a special transfer program, e.g., as a part of the operating system. In a step **310**, such special transfer program powers down the original, previous, core. In a step **312**, program control returns to the workload which begins executing at the point it reached when interrupted, e.g., step **215** (FIG. **2**).

Single instruction-set architecture (ISA) heterogeneous multi-core embodiments of the present invention are used to reduce overall average power consumption in an appliance. System software includes routines to evaluate the resources required by a running application for good performance. The system software dynamically chooses the one processor core that can best meet the present requirements while minimizing energy consumption. Alternatively, the system software dynamically chooses a next processor core that better meets the present requirements while minimizing energy consumption.

An analysis has shown that switching between five cores of varying performance and complexity can save, on an

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.