Cooperative Multithreading on Embedded Multiprocessor Architectures Enables Energy-scalable Design

Patrick Schaumont
EE Department
UCLA
CA 90095-1594
schaum@ee.ucla.edu

Bo-Cheng Charles Lai
EE Department
UCLA
CA 90095-1594
bclai@ee.ucla.edu

Wei Qin
ECE Department
Boston University
MA 02215
wqin@bu.edu

Ingrid Verbauwhede
EE Dept, UCLA, CA
and
ESAT, K.U. Leuven, BE
ingrid@ee.ucla.edu

ABSTRACT

We propose an embedded multiprocessor architecture and its associated thread-based programming model. Using a cycle-true simulation model of this architecture, we are able to estimate energy savings for a threaded C program. The savings are obtained by voltage- and frequency-scaling of the individual processors. We port a fingerprint minutiae detection application onto this architecture, and show the resulting performance on single-, dual-, and quad-processor configurations. The energy-scaled quad-processor version results in a 77 % energy reduction over the single-processor non-scaled implementation, at only a 2.2 % degradation in cycle count.

Categories and Subject Descriptors

C.3 [Special-purpose and Application-based Systems] Embedded Systems

General Terms

Design, Performance.

1. INTRODUCTION

The use of multiprocessors in system-on-chip context (MPSOC) enables applications that require high performance as well as low energy-consumption [8]. Indeed, under the assumption that an application can be partitioned equally over N processors, it can execute N times faster under constant clock frequency. We can also apply voltage scaling, and reduce the clock frequency. This results in reduced energy consumption, provided that the increased leakage power is appropriately addressed [12]. Recently, voltage-scaling techniques in the areas of scheduling [2][9] and circuit design [12][10] have been intensively studied. However, our work addresses the complementary field of system architecture and programming, and is an enabler for these scheduling- and circuit techniques.

This paper shows how a multiprocessor can be used as an energy-efficient replacement for a single processor. This is done by means of a thread-based programming model, and the use of voltage-scaling of individual processor units. Assume a single-thread application can be partitioned into multiple equal threads, each running on a separate processor. Rather than running on a single processor at nominal voltage, the application can then run on multiple processors at a slower clock and at a scaled voltage. This way, the overall energy-consumption of the application can be reduced without a significant penalty to the performance.

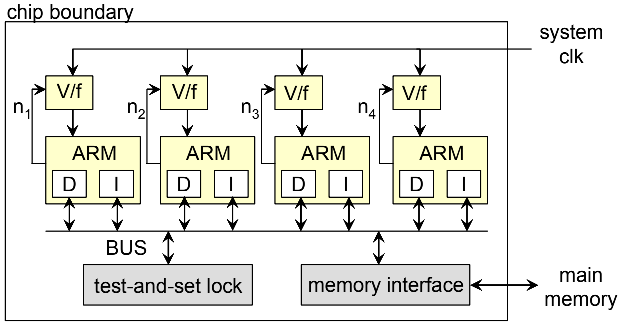


Figure 1. Energy-scaled MPSOC Architecture.

Compared to other voltage-scaled multiprocessors, such as MPCore from ARM [6], our approach is extremely lightweight. Our complete multithreading library including boot code fits in under 2Kbytes of object code. The second benefit we have is a high-performance cycle-accurate simulator of the multiprocessor. This way, we can obtain the energy-consumption of actual threaded C programs. In this paper, we present such a simulation result; we quantify the performance as well as the energy consumption of a fingerprint minutia detection application. These results will be obtained as follows. In section 2, we present the architecture of the multiprocessor. We discuss the mechanisms for process synchronization, as well as for energy scaling. In section 3, we discuss the programming model based on cooperative multithreading. In section 4, the ideas of sections 2 and 3 are brought together, by means of several examples including the fingerprint minutia detection application. In section 5, related work is reviewed, while section 6 puts the conclusions.

2. MULTIPROCESSOR ARCHITECTURE

An instance of our multiprocessor architecture is illustrated in Figure 1. The system includes four ARM cores, connected to a central bus. Each ARM has a 5-stage pipelined StrongARM micro-architecture. In addition to an execution unit, each ARM has a voltage/frequency scaling unit and local instruction- and data-caches. A bus connects the four ARM processors with a hardware test-and-set lock to support inter-process communication and synchronization, and a memory interface to access off-chip memory. We thus obtain a single-chip symmetric multiprocessing architecture. This allows the multiprocessor chip to be 'dropped-in' in place of a single-processor chip. Each of the four processors runs at only one fourth of the original single-processor clock frequency. Therefore, the total load on the memory interface will therefore not increase assuming an appropriate thread partitioning of the application. We now discuss the operation of the multiprocessor on-chip bus, the cache-coherency protocol, and the voltage/frequency (V/f) scaling units.

Bus Operation. The bus is organized as a multi-master bus. It supports two types of transactions: read- and write transactions. A transaction can be initiated by any master, but the bus will allow only a single transaction to proceed at a time. Bus access is granted using a round-robin protocol. The transaction time is three system clock cycles. When a bus slave is not ready to accept a transaction from a bus master, the transaction time will be extended until the slave is ready. The bus also contains a test-and-set lock, which supports atomic read-modify-write operations [1]. Memory-mapped semaphores are implemented in terms of this test-and-set lock.

Cache Coherency. A shared-memory multiprocessing model introduces a cache coherency issue [7]. This is resolved using a simple bus-snooping protocol. The caches implement a write-through policy, and all write-transactions over the system bus are observed by all caches in order to obtain coherency. The timings of our simulation model are as follows. Cache hits are completed in a single cycle. Memory accesses take 4 cycles for a word-write, while reading a complete cache line takes 24 cycles. The 4-cycle write time assumes a write buffer queue at the memory interface. A similar single-cycle write/merge buffer exists in the SA1100 implementation of the StrongARM.

Voltage/Frequency Scaling Units. The V/f units provide several discrete voltage/frequency operation modes for each of the processors. The bus and the bus interfaces of the caches always operate at nominal speed. Each of the V/f units is individually controlled out of embedded software. They are implemented by means of a 'power domain' technology, such as proposed in [10]. Combined voltage/frequency scaling is essential to achieve energy reduction. We use the energy model of Flautner [4] to estimate the relative energy savings that are obtained by scaling voltage and frequency. If a processor workload is divided in K pieces of scaled voltage v_i and frequency f_i, running T_i time units each, then the dynamic energy consumption for a single processor of capacitive load C_{eff} is equal to:

$$E_{AC,1} = C_{eff} \sum_{K} v_i^2 f_i T_i \qquad (1)$$

The dynamic energy of N processors is the sum of contributions of the individual processors:

$$E_{AC,N} = \sum_{N} E_{AC,j} \qquad (2)$$

Besides this dynamic component, there is also an important static component caused by leakage. This leakage can be minimized with circuit techniques, including multi- and variable-threshold CMOS (MTCMOS/TVCMOS), and adaptive body biasing [12]. The techniques in this paper are at the level of architecture and programming model, and can be combined with such circuit-level techniques.

Table 1. V/f pairs from some DVS projects.

Project	Pouwelse [14]	Flautner [4]
Processor	StrongArm	Xscale XSA
V/f high power (V/MHz)	1.65 / 251	1.5 / 773
V/f low power (V/MHz)	0.79 / 59	0.75 / 150
V^2f ratio (high/low)	18.5	20.6
f ratio (high/low)	4.25	5.15
switching time (us)	140	20

The V/f units apply different voltage/frequency pairs under control of embedded software. Some typical numbers of V/f pairs from literature are listed in Table 1.

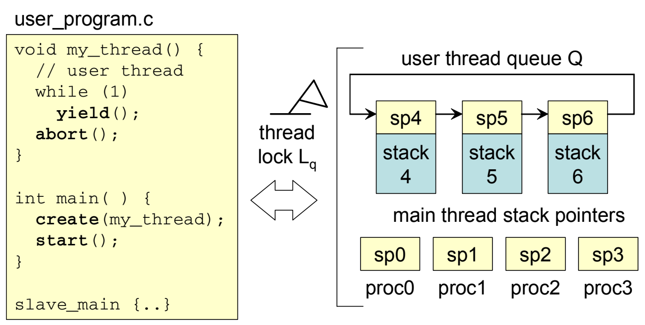
Cycle-accurate Performance Model. We have developed a cycle-accurate model of the multiprocessor architecture described above, based on a combination of SimIt-ARM [17] and GEZEL [18]. The simulation environment can be configured with different processor configurations, and achieves high simulation efficiency. We obtain 441000 cycles per second for a four-ARM-processor system on a 3GHz, 512MByte Fedora-2 PC. In our performance simulation model, we can dynamically change the clock frequency of the execution core of a processor. Our cycle-accurate system simulation works with divided clocks, in which processor clock rates have an integer relationship to the system clock rate. This is a flexible and easy-to-implement simulation mechanism. The clock division factors are programmed out of the ARM application by means of a system call. During simulation, the performance model also keeps track of the amount of clock cycles executed in each operation mode. This information, together with knowledge of valid V/f-pairs for a processor, can be used to evaluate the dynamic energy consumption according to Eq(1).

3. SOFTWARE PROGRAMMING MODEL

We next discuss the multiprocessor programming model and the cooperative multithreading system. Threads are a well-known software abstraction for task-level parallelism, appealing to a wide range of software developers. An important advantage of a threading approach to program a multiprocessor is that it can be done using a standard software tool-chain. The multiprocessor program is written in C and compiled using the arm-linux gcc 3.2.2 cross-compiler. Thread parallelism is supported by means of a software library.

Cooperative Multithreading. We have implemented a multiprocessor version of the QuickThreads cooperative multithreading library [11]. The data structures and procedural interface of the threading system are illustrated in Figure 2. Thread context information for user-defined threads is stored in a circular queue Q. The context information for each thread consists of a thread stack and a stack pointer — due to the cooperative scheduling, all context switches are done with known processor state, and processor registers (apart from the stack pointer) do not need to be stored. All processors use the same thread queue for a context switch. A central test-and-set lock L_q makes sure the thread queue can be modified by only a single processor at a time.

The processors use four routines to control thread creation, scheduling, and termination. New user threads are entered on the thread queue with `create`, which takes a function as argument. When a processor calls `start`, it will switch from the main thread (i.e. the `main` or `slave_main` thread of control) to the first user thread on the queue. If the thread



```
void my_thread() {
  // user thread
  while (1)
    yield();
  abort();
}

int main( ) {
  create(my_thread);
  start();
}

slave_main {..}
```

Figure 2. Multithreading data structures and their procedural interface.

queue is empty, the function returns immediately. The `yield` function puts a thread to the back of the user thread queue and transfers control to the entry on the queue front. A user thread terminates by calling `abort` or exiting the function that started the user thread.

The stack pointer of the main thread of each processor is not stored on the thread queue but in a global variable (`sp0` .. `sp3`). This way, the main threads can be tied to a single ARM processor, while the user threads can still migrate from one ARM processor to the next. These processor-specific main threads can then be used to unambiguously insert voltage-scaling commands, discussed next.

Voltage/frequency Scaling. The software abstraction of V/f scaling is a function call that selects a V/f mode. While voltage and frequency are adjusted independently in the implementation, they are always changed together according to a V/f pair table such as shown in Table 1. In the application software, the programmer selects a power mode by means of a system call. In our performance simulation, this system call is used to select the simulation frequency of the core, and to update energy-consumption statistics. The time required for a V/f mode to stabilize depends on the implementation of the V/f scaling technology. The use of clock division, in combination with power switch tiles [10] gives a few discrete voltage/frequency settings with very low switching time.

A V/f-scaled multiprocessor has multiple clock frequencies. We will use the term processor clock cycles to indicate clock cycle counts of individual processors, and the term system clock cycles to indicate clock cycle counts at the clock input of the multiprocessor system.

Discussion. Multithreading ensures scalable parallelism on top of a multiprocessor system. A program with 5 threads, for example, will run on a single processor but also on a quad-processor. A central scheduler is not required because of the cooperative multithreading, which benefits the multiprocessor fault-tolerance. The software development approach is an incremental evolution of standard practice. It does not require a radical rethinking of the system-on-chip development process. An application, originally running as a sequential C program, can be incrementally partitioned into multiple threads, and run on the multiprocessor gradually exposing parallellism.

4. RESULTS

In this section we consider multiprocessor mapping of a fingerprint minutiae detection algorithm onto the multiprocessor architecture. Fingerprint minutiae detection

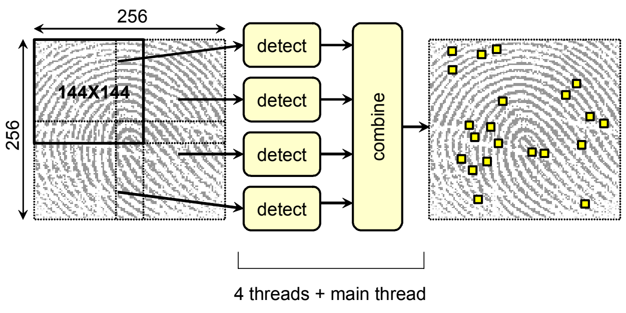


Figure 3. Parallel Minutiae Detection.

[15] is an image-processing operation taken out of the biometrics domain. We use single-processor, dual-processor and quad-processor configurations. Each time, we consider two operation modes for each core, a high-power mode and a low-power mode. We use V/f pairs as shown in Table 2.

Table 2. Processor V/f scaling for experiments.

	high-power (H)	low-power (L)
processor clk/ system clk	1/1	1/5
Relative power (V^2f) [W/μF]	683	37

Fingerprint minutiae detection is traditionally run on fairly high-powered processors, such as on PC's. However, an important class of applications, including portable electronic keys, require energy-efficient implementation of minutiae detection. The reference algorithm used in this paper is a fixed-point version of the NIST fingerprint software. The algorithm works on grayscale images of 256 by 256 pixels. The detection of minutia is a computation-intensive operation, and involves multiple phases of image processing. On a single ARM processor, the sequential C version of this algorithm takes over 200 million clock cycles to complete, to detect typically 50 to 100 minutiae on a fingerprint image.

Task partitioning. For the development of a task-level parallel algorithm of minutiae detection, we make use of the property that minutiae detection is a local image processing algorithm. Indeed, a fingerprint ridge ending or bifurcation can be detected on a relatively small area of a fingerprint. This leads to a task partitioning as illustrated in Figure 3. The original fingerprint image contains 256 by 256 pixels. Rather than running minutiae detection over the complete image, we detect minutiae in a smaller portion of the original image, containing 144 by 144 pixels. The size of 144 by 144 pixels has been chosen to guarantee an overlap between different subregions. The fingerprint minutiae detection algorithm thus decomposes into four different tasks, that each can be mapped to a different thread. The detection algorithm accounts for the majority of the clock cycles. In fact, the task partitioning shown in Figure 3 results in an almost perfect distribution of workloads. Over 95% of the total clock cycles are contained in the four detection sub-tasks.

Performance/energy comparison. Figure 4 shows performance/energy for single-, dual-, and quad-processor architectures under different voltages. The labels on the X-axis indicate the configuration. For example, 2_HL represents a dual-processor architecture with one processor in high-power mode and another one in low-power mode. Figure 4 clearly points out the relevance of voltage-scaled multiprocessor architectures. Two configurations in particular improve on the single-processor nominal case (1_H). The first of them is

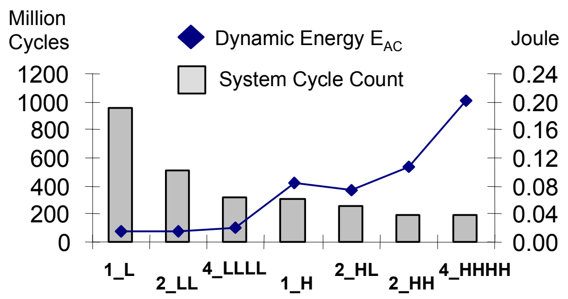


Figure 4. Impact of V/f scaling on Performance/Energy.

2_HL, a dual-processor implementation. Compared to a single-processor nominal version (1_H), this architecture offers lower energy (12%) as well as lower execution time (16%). The second configuration is that of a quad-processor at low-power (4_LLLL). It runs only 2.2 % slower than the single processor version at nominal voltage (1_H), yet it gives more than 77% reduction in energy consumption. Figure 4 shows a smooth tradeoff between performance and energy consumption. In fact, by a simple extension of the V/f-scaling units and the addition of a central controller, a processor could be completely turned off as well. In that case, a quad-processor configuration is able to implement all the design points for a single-processor and a dual-processor.

5. RELATED WORK

Our work is complementary to the existing research on scheduling for voltage-scaling. Voltage-scaling scheduling strategies have been intensively researched, for example in the work of Jejurikar [9] or Andrei [2]. These strategies are however based on abstract task models, and none of them use cycle-true instruction-set simulation to validate the results. An obvious argument for on-chip multiprocessing has been cycle-count performance. For example, Petrot proposes a pthread-based programming model for an on-chip MIPS multiprocessor [13]. Forsell describes the parallel random-access machine (PRAM), as a class of multi-threaded shared-memory architectures. These models focus on performance rather than on energy-efficiency. ARM's MPCore [7] proposes voltage-scaled multiprocessing. In contrast to our approach, they use a heavy kernel-thread programming model, based on Linux SMP; and they use symmetric speed scaling, in which the voltage and frequency of all cores is varied together. Simultaneous multithreading [3] and hyper-threading implement support for task-level parallelism at the processor level. However, such processors consist of several tightly integrated pipelines, that cannot be voltage-scaled in the same way as can be done with multiprocessors. Task-level parallelism is a newcomer in the area of on-chip parallellism. Indeed, over the past years, instruction-level parallelism has received more attention, both academically as well as commercially. This has resulted in commercially available ASIP (such as Xtensa from Tensilica) and VLIW architectures (such as Optimode from ARM). An early task-level system programming model was proposed by Improv [16].

6. CONCLUSIONS

We proposed the combination of an architecture and a programming model for energy-scalable on-chip implementation of C programs. The strong points of our work are a cycle-accurate performance model that allows us to evaluate actual applications; a programming model that is familiar to a large group of programmers; and an energy-scaling technology that is easy to understand and realistic to implement. Currently we are investigating the detailed implementation of the on-chip bus, the usage of new forms of bus communication and the influence of advanced energy scaling technology.

7. ACKNOWLEDGEMENTS

This work was possible thanks to the support of SRC (Grant 2003-HJ-1116) and the support of NSF (Grant CCR 0310527).

8. REFERENCES

[1] G. Andrews, "Concurrent programming - principles and practice", 102—105, Benjamin Cummings Publ. 1991.

[2] A. Andrei, M/ Schmitz, P. Eles, Z. Peng, B. Al-hashimi, "Overhead-Conscious Voltage Selection for Dynamic and Leakage Energy Reduction of Time-Constrained Systems," Proc. DATE 2004, 518—523.

[3] S. Eggers, J. Emer, H. Levy, J. Lo, R. Stamm, D. Tullsen, "Simultaneous multithreading: a platform for next-generation processors," IEEE Micro, 1997, 17(5):12—19.

[4] K. Flautner, S. Reinhardt, T. Mudge, "Automatic Performance Setting for Dynamic Voltage Scaling," Wireless networks, 2002, 8(5), 507—520.

[5] M. Forsell, "A Scalable high-performance computing solution for networks on chips,", IEEE Micro, 2002, 22(5):46—55.

[6] J. Goodacre, "Challenges in programming multiprocessor platforms," 4th International seminar on Application-Specific MPSOC, France, 2004.

[7] J. Hennessy, D. Patterson, "Computer Architectures: A quantitative approach," Ch. 6.3, MKP Publishers, 2002.

[8] A. Jerraya, W. Wolf, "Multiprocessor Systems-on-Chips," Morgan Kaufmann, Sept 2004, ISBN 0-12-385251-X.

[9] R. Jejurikar, C. Pereira, R. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," Proc. DAC 2004:275—280.

[10] J. Rabaey, "Power Management in Wireless SOCs," 4th International seminar on Application-Specific MPSOC, France, 2004.

[11] D. Keppel, "Tools and Techniques for Building Fast Portable Threads Packages," UWCSE 93-05-06, U. Washington, 1993.

[12] S. Martin, K. Flautner, T. Mudge, D. Blaauw, "Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Lower Power Microprocessors under Dynamic Workloads," Proc. ICCAD 2002:721—725.

[13] F. Petrot, P. Gomez, "Lightweight implementation of the POSIX Threads API for an On-chip MIPS Multiprocessor with VCI Interconnect," Proc. 2003 DATE:51—56.

[14] J. Pouwelse, K. Langedoen, H. Sips, "Application-directed voltage scaling," IEEE Trans. on VLSI Systems, 11(5):812—826.

[15] S. Yang, K. Sakiyama, I. Verbauwhede, "A compact and efficient fingerprint verification system for secure embedded systems," 37th Asilomar Conference, Nov 2003:2058—2062.

[16] C. Ussery, " Method of generating application specific integrated circuits using a programmable hardware architecture," US. Pat. 6,075,935, 12/1/1997.

[17] W. Qin, S. Malik. Flexible and Formal Modeling of Microprocessors with Application to Retargetable Simulation, 2003 Design Automation and Test in Europe, March 2003, 556—561.

[18] P. Schaumont, I. Verbauwhede, "Interactive cosimulation with partial evaluation," 2004 Design Automation and Test in Europe, Fe-bruary 2004, 642—647.