# A Quantitative Analysis of Reconfigurable Coprocessors for Multimedia Applications

Takashi Miyamori
System ULSI Engineering Laboratory
TOSHIBA Corporation, JAPAN
miyamori@sdel.toshiba.co.jp

Kunle Olukotun
Computer Systems Laboratory
Stanford University
kunle@ogun.stanford.edu

## Abstract

*Recently, computer architectures that combine a reconfigurable (or retargetable) coprocessor with a general-purpose microprocessor have been proposed. These architectures are designed to exploit large amounts of fine grain parallelism in applications. In this paper, we study the performance of the reconfigurable coprocessors on multimedia applications. We compare a Field Programmable Gate Array (FPGA) based reconfigurable coprocessor with the array processor called REMARC (Reconfigurable Multimedia Array Coprocessor). REMARC uses a 16-bit simple processor that is much larger than a Configurable Logic Block (CLB) of an FPGA. We have developed a simulator, a programming environment, and multimedia application programs to evaluate the performance of the two coprocessor architectures. The simulation results show that REMARC achieves speedups ranging from a factor of 2.3 to 7.3 on these applications. The FPGA coprocessor achieves similar performance improvements. However, the FPGA coprocessor needs more hardware area to achieve the same performance improvement as REMARC.*

## 1 Introduction

As the use of multimedia applications increases, it becomes important to achieve high performance on algorithms such as video compression, decompression, and image processing with general-purpose microprocessors. This has motivated the recent addition of multimedia instructions to most general-purpose microprocessor ISAs [1–3]. These ISA extensions work by segmenting a conventional 64-bit datapath into four 16-bit or eight 8-bit datapaths. The multimedia instructions exploit fine grain SIMD parallelism by operating on four 16-bit or eight 8-bit data values. However, a 64-bit datapath limits the speedups to a factor of four or eight even though many multimedia applications have much more inherent parallelism.

Computer architectures that connect a reconfigurable coprocessor to a general-purpose microprocessor have been proposed [4–9]. The advantage of this approach is that the coprocessor can be reconfigured to improve the performance of a particular application. All of these proposed architectures use field programmable gate arrays (FPGAs) for the reconfigurable hardware. We refer to this coprocessor as an "*FPGA coprocessor*" in this paper. The FPGA architecture, which has narrow programmable logic blocks and programmable interconnection network, provides great flexibility for implementing application specific hardware. However, the rich programmable interconnection comes at the price of reduced operating frequency and logic density.

Array processors, such as general-purpose systolic array processors, wavefront array processors [10], PADDI-2 [11], and MATRIX [12], are other reconfigurable architectures. These processors have 8-bit or 16-bit datapaths and each programmable logic block has an 8 to 32-entry instruction RAM that makes it easy to support multiple functions. Because multimedia (or DSP) applications predominantly manipulate 8-bit or 16-bit data values, these architectures work very well on these applications. Recently, we proposed a new array processor architecture called REMARC (Reconfigurable Multimedia Array Coprocessor)[13]. REMARC is a reconfigurable coprocessor that is tightly coupled to a main RISC processor and consists of a global control unit and 64 16-bit simple processors called nano processors.

Both the FPGA coprocessor and REMARC are not limited to SIMD parallelism that can be exploited by multimedia extensions such as the Intel MMX[2]. They can exploit various kinds of fine grain parallelism in multimedia applications. Using more processing resources, they can achieve higher performance than the multimedia extensions. To understand how these two coprocessor architecture compare, in this paper we evaluate the cost and performance of these architectures. The architecture of FPGA coprocessors are still in flux, so we evaluate the performance of the FPGA coprocessor with a varying number of CLBs and vary the cycle time of the FPGA coprocessor from 1x to 10x that of the main processor. For the performance evaluation, we use detailed simulators and two realistic application programs, DES encryption and MPEG-2 decoding. We also estimate the chip sizes of processors with REMARC and the FPGA coprocessor and compare their performance when the same die size is used for both architectures.

The rest of this paper is organized as follows. In Section 2, we describe the reconfigurable coprocessor architectures, both REMARC (array based) and FPGA based. In Section 3, we show the results of our performance evaluation. In Section 4, we estimate chip sizes of processors

with REMARC and the FPGA coprocessor. Finally, we conclude in Section 5.

# 2 Reconfigurable Coprocessor Architecture

## 2.1 Architecture Overview

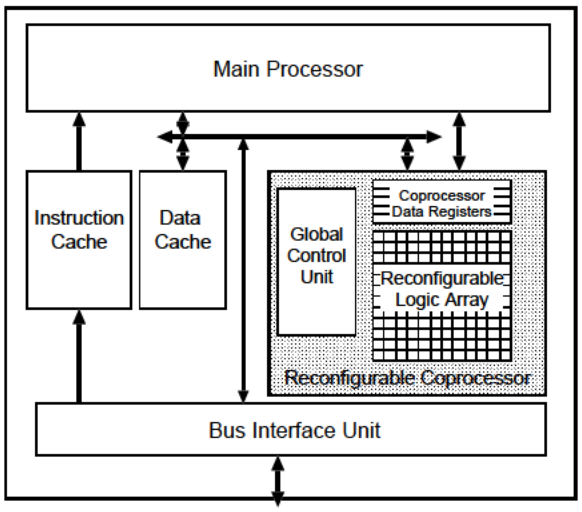


Figure 1: Block Diagram of a Microprocessor with Reconfigurable Coprocessor

Figure 1 shows a block diagram of a microprocessor which includes a reconfigurable coprocessor. The reconfigurable coprocessor consists of a global control unit, coprocessor data registers, and a reconfigurable logic array. Recently, we proposed the REMARC architecture which includes an 8x8 16-bit processor (nano processor) array as its reconfigurable logic array[13]. The other reconfigurable coprocessor that we consider in this paper, the FPGA coprocessor, uses FPGAs for the reconfigurable logic array. The global control unit controls the execution of the reconfigurable logic array and the transfer of data between the main processor and the reconfigurable logic array through the coprocessor data registers.

We use the MIPS-II ISA [14] as the base architecture of the main processor. The MIPS ISA is extended for the REMARC and the FPGA coprocessor using the instructions listed in Table 1. The main processor issues these instructions to the reconfigurable coprocessor which executes them in a manner similar to a floating point coprocessor. Unlike a floating point coprocessor, the functions of reconfigurable coprocessor instructions are configurable (or programmable) so that they can be specialized for specific applications.

The configuration instructions, *rcon*, *rgcon*, or *rncon*, download the configuration data from memory and store them in the reconfigurable coprocessor. The start address of the configuration data is specified by the value of the source register (*src*). The *rex* instruction starts execution of a reconfigurable coprocessor instruction. The sum of the *offset* field and the *base* register specifies one of the operations to execute. The *lduc2* and *sduc2* instructions are load and store coprocessor instructions which transfer double word (64-bit) data between memory and the reconfigurable coprocessor data registers. The *mfc2* and *mtc2* instructions transfer word (32-bit) data between the general-purpose registers (integer registers) in the main processor and the reconfigurable coprocessor data registers. The *cfc2* and *ctc2* instructions transfer data between the integer registers and the reconfigurable coprocessor control registers.

| | |
|---|---|
| *rcon* | *src (rncon or rgcon)* |
| *rex* | *cop2_reg, offset(base)* |
| *lduc2* | *cop2_reg, offset(base)* |
| *sduc2* | *cop2_reg, offset(base)* |
| *mtc2* | *cop2_reg, src* |
| *mfc2* | *cop2_reg, dst* |
| *ctc2* | *cop2_reg, src* |
| *cfc2* | *cop2_reg, dst* |

Table 1: New Instructions Used in Reconfigurable Coprocessors

The reconfigurable coprocessors do not have a direct interface to the data cache or memory. The main processor has to set the input data to the coprocessor data registers using *lduc2* and *mtc2* instructions before execution of *rex* instructions. Then, the reconfigurable coprocessor reads the input data, executes the operations, and stores the results into the coprocessor data registers. Finally, the main processor reads the results using *sduc2* and *mfc2* instructions.

## 2.2 Pipeline Organization

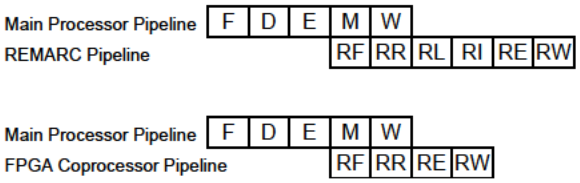


Figure 2: Pipeline Organization of Reconfigurable Coprocessor

The pipeline for REMARC, the FPGA coprocessor, and the main processor are shown in Figure 2. The main processor pipeline is similar to the MIPS R3000 and the MIPS R5000 and consists of five stages: Instruction Fetch (F), Instruction Decode (D), Execution (E), Memory Access (M), Register Write-back (W). The reconfigurable coprocessor pipelines are independent of the main processor pipeline; therefore, the main processor can execute concurrently with the reconfigurable coprocessors.

The REMARC pipeline starts from the M stage of the main processor and has the following six stages:

**RF :** An instruction of the global control unit is fetched.
**RR :** The REMARC data registers are read.
**RL :** The data are aligned or "unpacked".
**RI :** The instructions of the nano processors are fetched.

**RE :** The nano processors execute the instructions.

**RW :** The executed results are packed and stored into the REMARC data registers.

The FPGA coprocessor pipeline has four stages as follows:

**RF :** The sequencer of the global control unit starts its execution.

**RR :** The coprocessor data registers are read.

**RE :** The reconfigurable logic array starts execution.

**RW :** The execution results are stored into the coprocessor data registers.

The RL and RI stages are unnecessary in the FPGA coprocessor because load alignment or unpack operations are realized directly in the FPGA array and FPGAs do not have instructions to fetch and execute.

## 2.3 REMARC Architecture

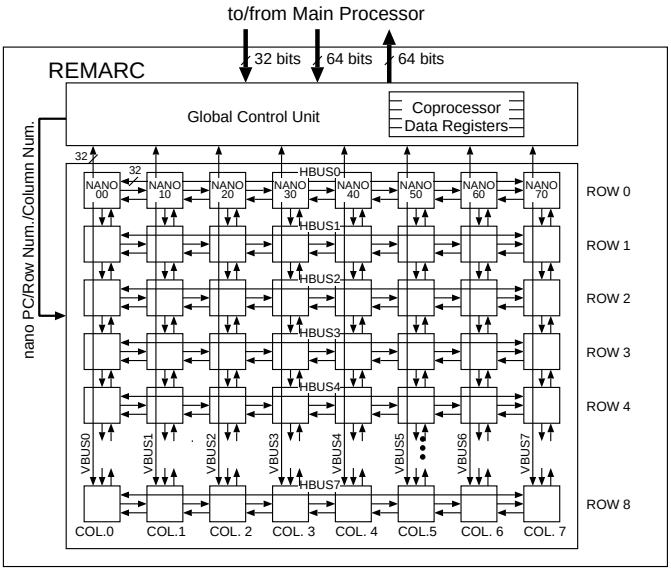


Figure 3: Block Diagram of REMARC

In this section, we briefly describe the REMARC architecture; more information can be found in [13]. Figure 3 shows a block diagram of REMARC. REMARC's reconfigurable logic is composed of an 8x8 array of the 16-bit processors, called nano processors. The execution of each nano processor is controlled by the instructions stored in the local instruction RAM (nano instruction RAM). However, each nano processor does not directly control the instructions it executes. Every cycle the nano processor receives a PC value, "nano PC", from the global control unit. All nano processors use the same nano PC and execute the instructions indexed by the nano PC in their nano instruction RAM.

Figure 4 shows the architecture of the nano processor. The nano processor contains a 32-entry nano instruction RAM, a 16-bit ALU, a 16-entry data RAM, an instruction

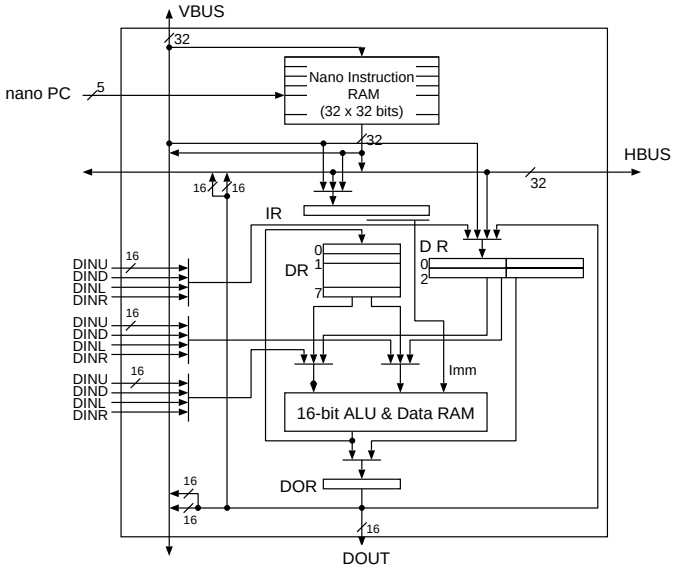


Figure 4: Nano Processor Architecture

register (IR), eight 16-bit data registers (DR), four 16-bit data input registers (DIR), and a 16-bit data output register (DOR).

Each nano processor can use the DR registers, the DIR registers, and immediate data as the source data of ALU operations. Moreover, it can directly use the DOR registers of the four adjacent nano processors (DINU, DIND, DINL, and DINR) as the source.

The nano processors are also connected by the 32-bit Horizontal Buses (HBUSs) and the 32-bit Vertical Buses (VBUSs). Each bus operates as two 16-bit data buses. The 16-bit data in the DOR register can be sent to the upper or lower 16 bits of the VBUS or the HBUS. The HBUSs and the VBUSs allow data to be broadcast to the other nano processors in the same row or column. These buses can reduce the communication overhead between processors separated by long distances.

The DIR registers accept inputs from the HBUS, the VBUS, the DOR, or the four adjacent nano processors. Because the width of the HBUS and the VBUS is 32 bits, data on the HBUS or the VBUS are stored into a DIR register pair, DIR0 and DIR1, or DIR2 and DIR3. Using the DIR registers, data can be transfered between nano processors during ALU operations.

It takes a half cycle to transfer data using the VBUSs or HBUSs. It should not be a critical path of the design. Other operations, except for data inputs from nearest neighbors, are done within the nano processor. Because the width of a nano processor's datapath is only 16 bits, which is a quarter of those of the general purpose microprocessors, this careful design does not make REMARC a critical path of the chip.

## 2.4 FPGA Coprocessor Architecture

The reconfigurable logic array of the FPGA coprocessor is composed of configurable logic blocks (CLBs). Each CLB

| | REMARC | FPGA Coprocessor |
|---|---|---|
| Processor Size | Large (16-bit datapath) | Small (two 4-1 LUTs) |
| Num. of Procs. | Small (64 processors) | Large (500 – 5000 CLBs) |
| Execution control | Instruction | Hardwired |
| Communication | Controlled by instruction | Configured by switch matrix |
| Interconnection | 4 neighbors, VBUS, and HBUS | Short wires and long wires |
| Cycle Time | Tcpu | 1x, 2x, 5x, 10xTcpu |

Table 2: Coprocessor Architecture Comparison

is equal to a CLB of the Xilinx 4000 series. The CLB includes two 4-1 lookup tables (LUTs) and two flip-flops.

We do not fix the number of the CLBs in the FPGA coprocessor. Instead, we evaluate the performance of the FPGA coprocessor using a varying number of CLBs. The cycle time of the FPGA coprocessor is also parameterized. Cycle times of current FPGA systems are longer than those of the microprocessors by factors of five to ten. For instance, FPGA systems usually operate at 30 to 100 MHz while state-of-the-art microprocessors operate at more than 400 MHz. However, the recently proposed reconfigurable coprocessor Garp [8] aims to operate at the same operating frequency as its main processor. Therefore, we assumed the cycle time of the FPGA coprocessor could be 5x or 10x that of the main processor for current FPGA architectures and 1x or 2x for future FPGA architectures.

### 2.5 Coprocessor Architecture Comparison

Table 2 summarizes the comparison of the two reconfigurable coprocessor architectures. REMARC has larger processing elements, the nano processors, than the FPGA coprocessor. However, the number of nano processors is less than that of CLBs in the FPGA coprocessor. In REMARC, both the execution and data transfer are controlled by instructions, while these are controlled by hardwired logic in the FPGA coprocessor. REMARC has limited hardware interconnections. Each nano processor has direct inputs from the four nearest neighbors and it is connected by two 32-bit data buses. The FPGA coprocessor has more flexible hardware interconnections which are configured in a bit-wise fashion.

We assume that REMARC will operate at the same frequency as the main processor. The cycle times of the FPGA coprocessor ($Tfpga$) are varied. We evaluate the FPGA coprocessor performance, assuming Tfpga values that are 1x, 2x, 5x, and 10x the CPU cycle time ($Tcpu$).

## 3 Performance Evaluation

### 3.1 Simulation Methodology

We developed the reconfigurable coprocessor simulator using the SimOS simulation environment [15] . SimOS models the CPUs, memory systems, and I/O devices in sufficient detail to boot and run a commercial operating system. As a base CPU simulation model, we used the "*MIPSY* " which models a simple single issue RISC processor similar to the MIPS R3000.

REMARC functions are added to the MIPSY model. The latency of a reconfigurable execution (*rex*) instruction is the sum of the number of the executed global instructions and the pipeline latency (5 cycles). If a following instruction attempts to read the result of a *rex* instruction, the pipeline will stall for the cycles of the *rex* instruction's latency (Data Dependency). Furthermore, a following *rex* instruction will stall if a previous *rex* instruction is still executing (Resource Conflict).

We evaluate the execution time of the FPGA coprocessor by changing the latencies of the *rex* instructions. First, we estimate the number of execution cycles of the *rex* instructions based on the FPGA delay model. In this model, two sequences can be executed within one FPGA cycle:

- One stage of interconnection by long or short wire and one stage of function not using the carry chain
- One stage of interconnection by short wire and any one stage of function

This model is almost the same as the Garp's delay model [8] and the XC4000's medium frequency design which will operate at about 70 MHz [16]. Then, we normalize the number of execution cycles based on the CPU cycle time, assuming the FPGA cycle time is 1x, 2x, 5x, or 10x the CPU cycle time. Finally, we use this estimated cycle count of the *rex* instruction in the simulator.

We also developed simulators which can execute multimedia instructions similar to the Intel MMX instruction set extensions [2].

To make the comparison fair, the same application source codes are used for the evaluation except for the use of the multimedia instructions or the augmented coprocessor instructions. The memory system parameters used commonly through the performance evaluations are found in Table 3. We used the "gcc" compiler with the "-O2" optimization option. This option executes most global compiler optimizations except for loop unrolling and function inlining.

| | |
|---|---|
| I-cache | 32 K bytes, 2-way set. |
| D-cache | 32 K bytes, 2-way set. |
| L2 cache | 256 K bytes, 2-way set. |
| L1 miss penalty | 5 cycles |
| L2 miss penalty | 50 cycles |

Table 3: Memory System Parameters

## 3.2 DES Encryption

The Data Encryption Standard (DES) is one of the most important encryption algorithms and has been a worldwide standard for over 20 years. It is widely used to provide secure communication over the Internet. DES is also a good application for reconfigurable processors because it has a lot of fine-grained parallelism in the form of bit-level irregular data movements which make software implementation on conventional microprocessors difficult and inefficient.

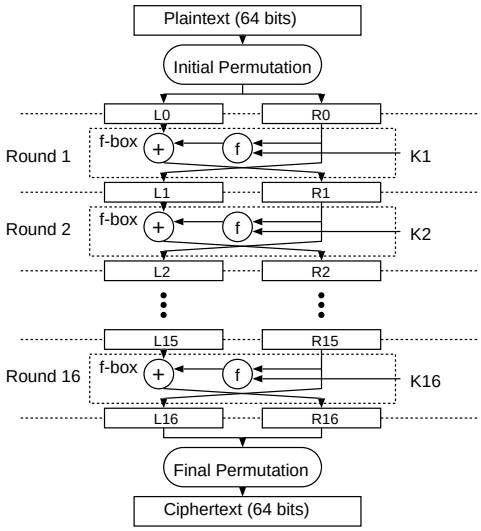


Figure 5: DES Encryption Algorithm

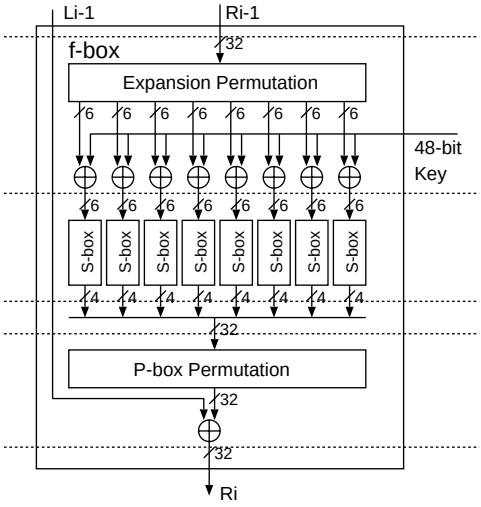


Figure 6: DES f-box Algorithm

Figure 5 shows an outline of the DES algorithm. DES takes as input 64-bit plaintext. After the initial permutation, there are 16 rounds of "f-box" operations, as shown in Figure 6, including expansion permutation, XOR with the key, S-box table lookup, P-box permutation, and XOR with the result of the previous round. The final permutation is performed on the 64-bit result of the sixteenth round.

We used the DES encryption program [17] based on the Electronic Codebook (ECB) mode. Although the ECB mode is less secure than the Cipher Block Chaining (CBC) mode, it is more commonly used and its operation can be pipelined.

### 3.2.1 DES Implementation on REMARC

We decided to divide the algorithm between the main processor and REMARC. The initial permutation and the final permutation are executed by the main processor and the 16 rounds of f-box operations are executed by REMARC. Each row of nano processors executes two f-box operations. For instance, eight nano processors in the row 0 execute the first and second rounds, the row 1 execute the third and fourth rounds, and so on.

| Operation | CPU Cycles |
|---|---|
| Data Load | 2 |
| Exp. Permutation | 6 ( 3 x 2 iter.) |
| Key XOR | 2 ( 1 x 2 iter.) |
| S-box | 12 ( 6 x 2 iter.) |
| P-box Permutation | 22 ( 11 x 2 iter.) |
| Left XOR | 4 (2 x 2 iter.) |
| Data Transfer | 3 |
| Total | 51 |

Table 4: Execution Cycle Breakdown of Two f-box Operations on REMARC

Table 4 is the execution cycle breakdown of the two f-box operations implemented by the eight nano processors in each row. The 16 f-box operations are pipelined into 8 stages by the eight rows of the nano processor array. REMARC can generate a result of the 16 f-box operations every 51 cycles. As Table 4 shows, because of the limited interconnection of REMARC, more than half (28 cycles) of the execution time are used for the expansion permutation and the P-box permutation.

### 3.2.2 DES Implementation on the FPGA Coprocessor

First, we estimate the latency and the throughput of one f-box operation. The expansion permutation and the XOR operation with the keys can be executed in one cycle because it can be implemented by long wire and simple logic. The S-box table lookup requires two cycles to execute because it consists of LUTs and MUXs. The P-box permutation and the XOR operation can be executed in one cycle. The total latency of the f-box operation is four cycles, and it can be fully pipelined. Therefore, the maximum throughput of the f-box operation is one FPGA cycle.

We assume three distinct cases, implementing one f-box, 16 f-boxes, and all of the DES encryption algorithm including 16 f-boxes, the initial permutation, and the final permutation.

In the case of one f-box implementation, because each f-box operation takes as input the result of the previous f-box operation, the f-box operations cannot be pipelined.

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.