ELM: Efficient Low-power Microprocessor

Efficient programmable fabrics for embedded applications

Professor William J. Dally (PI) James Balfour James Chen
Curt Harting Jongsoo Park David Sheffield

Stanford University
Electrical Engineering

Concurrent VLSI
Architecture Group
Computer Systems Lab

The Problem

Embedded applications are becoming more **complicated** and computational **demanding**.

- ASICs are too inflexible
- Processors are too inefficient

Algorithms and standards are very complex and change rapidly. ASIC development takes too long and is too costly.

Programmable processors and DSPs spend most of their energy moving data and instructions, instead of on computation. Their serial execution model and generic instruction set poorly exploit the parallelism in embedded applications.

A new **efficient embedded architecture** is needed to address this divergence. It must scale from personal handsets (GOPS) to cellular base stations (TOPS) while providing high energy efficiency.

Closing the ASIC Gap

Current Microprocessors and DSPs are **30-100x less efficient** than ASICs in ops/Watt or ops/mm².

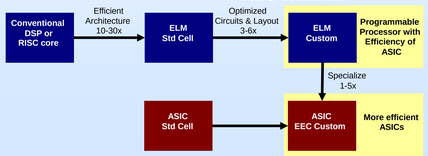
Microprocessors/DSP

- <10MOPS/mW
- ~ 0.1GOPS/$
- <10GOPS peak
- 1M$ programming cost
- Programmable

ASIC/ASSP

50-200 MOPS/mW
2-10 GOPS/$
Up to 1000GOPS peak
15M-25M$ design cost
Fixed function

We plan to close this gap by addressing inefficiencies in **architecture**, using optimized **circuits**, and utilizing a productive **programming system** to generate an efficient embedded solution that scales from handsets to base stations.



Application-specialized ELM cores can be extracted to implement **more efficient ASIC designs**, with **faster design times**, for **lower cost**.

Energy of Operations

Datapath Operations		Relative Energy
32-bit addition	520 fJ	1×
16-bit multiply	2,200 fJ	4.2×
32-bit pipeline register	330 fJ	0.63×
EP		
XRF – 32 entries 2R+2W		
32-bit read	200 fJ	0.38×
32-bit write	370 fJ	0.71×
GRF – 8 entries 2R+2W		
32-bit read	103 fJ	0.20×
32-bit write	120 fJ	0.23×
ARF – 8 entries 2R+2W		
32-bit read	103 fJ	0.20×
32-bit write	120 fJ	0.23×
ORF – 4 entries 2R+2W		
32-bit read	55 fJ	0.11×
32-bit write	95 fJ	0.48×
IRF – 64 entries 1R + 1W		
64-bit read	580 fJ	1.1×
64-bit write	1,150 fJ	2.2×
Ensemble Memory 256 × 64-bits		
32-bit load	1,400 fJ	2.7×
32-bit store	2,430 fJ	4.7×
Execute an add instruction	**1,635 fJ**	3.1×

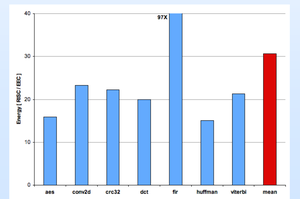
Conventional RISC		
Register File – 32 entries 4R+2W		
32-bit read	250 fJ	0.48×
32-bit write	470 fJ	0.90×
Data Cache – 256 × 64-bit 4-way set associative		
32-bit load	3,540 fJ	6.8×
32-bit store	3,530 fJ	6.8×
miss	1,410 fJ	2.7×
Instruction Cache – 256-entry 4-way set associative		
64-bit fetch	3,500 fJ	6.8×
miss	1,410 fJ	2.7×
128-bit refill	9,710 fJ	19×
Filter Cache – 64-entry direct-mapped		
64-bit fetch	990 fJ	1.9×
miss	430 fJ	0.82×
128-bit refill	2,560 fJ	4.9×
Filter Cache – 64-entry fully-associative		
64-bit fetch	1,320 fJ	2.5×
miss	980 fJ	1.9×
128-bit refill	2,610 fJ	5.0×
Execute an add instruction	**5,320 fJ**	10.2×

Evaluation

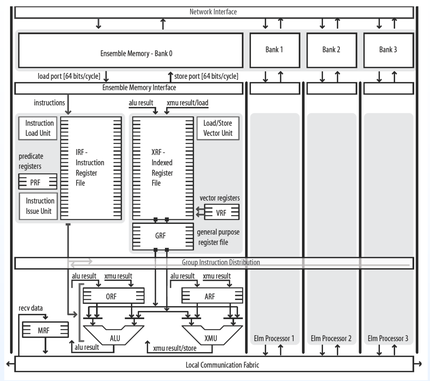
We have compared a single Ensemble Processor (EP) to the LEON2 SPARC v8 embedded RISC core, demonstrating a 30x efficiency improvement. We also found that ELM efficiency for embedded kernels comes within 2-5x of an ASIC implementation.

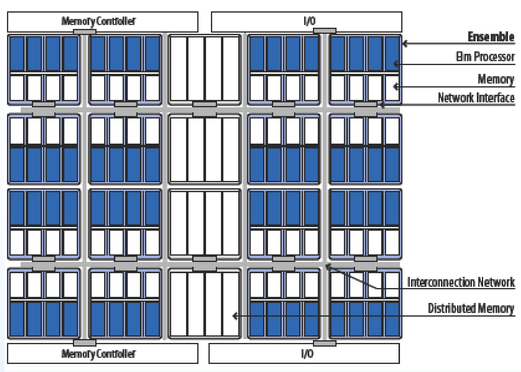
Recently, we have focused on refining and analyzing specific aspects of ELM. Studies have included finding the best configurations for the instruction and data registers, the benefits of custom circuits, and how compiler algorithms impact energy use.

Future work includes refinement of the global architecture, programming system, and the fabrication of a chip.



Architecture





System Level

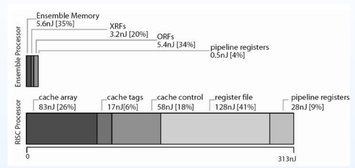
- Small in-order Ensemble Processors (EPs)
- Four EPs and small SRAM array form an Ensemble
- Chip is comprised of distributed memory tiles and Ensembles
- Intra-Ensemble communication occurs either via the Ensemble memory or message registers
- Inter-Ensemble communication is most efficiently done via software controlled data streams across the interconnection fabric
- Streaming operations allow for latency hiding and code size reduction
- Allowing for software control both guarantees real time constraints and minimizes wasted data movement energy

Datapath

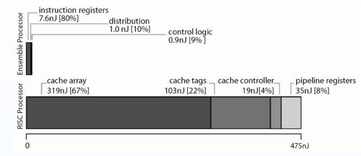
- Each EP has two pipelines: address and arithmetic
- The address pipeline is responsible for issuing loads and stores, as well as performing basic arithmetic operations
- The arithmetic pipeline is used to perform operations on program data. It includes a shifter, multiplier, adder, and zero's counter
- Each of these pipelines have a 4 entry SRAM (ARF/ORF, respectively) that can be accessed in the execute cycle
- Bypassing is explicitly managed by software
- Mechanisms for auto-updating counters to reduce loop overheads

Data Supply

- RISC processors load data from a large, tagged reactive cache into a large register file
- ELM contains a distributed register hierarchy comprised of small (4-8 entry) SRAM arrays
- These arrays are physically and temporally near the functional units
- Backed by the tag-less Ensemble memory



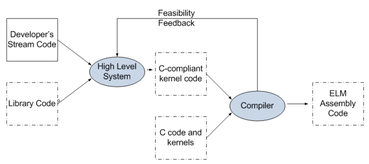
Instruction Supply



- RISC processors fetch instructions out of a tagged, reactive L1 loop cache
- ELM executes out of a 64 entry tag-less, software controlled register file (IRF)
- Software has specialized fetch instruction to bring code blocks into the IRF

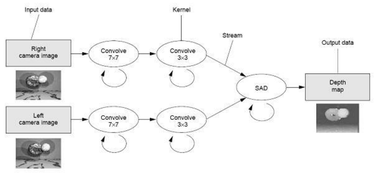
Programming System

Giving algorithm designers a productive implementation flow



- The ELM programming system works with the underlying compiler to ensure that kernels can be executed in the allotted amount of time
- It also works with the programmer, providing feedback on the feasibility of partitioning schemes
- A sample program representation can be seen below

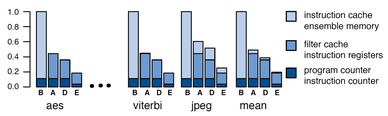
- Programmers write in a high-level language, providing hints about kernel boundaries and streaming operations
- An ELM program parses this program into kernels based on the programmer's division of tasks
- This program can split kernels, merge kernels, and setup buffers to find a code mapping onto the compute fabric that meets the programmer's real time constraints



Compiler

Managing the memory hierarchy to reduce energy

- One of the novel primary tasks of the compiler is to schedule instructions for the IRFs.
- Fetch hoisting and minimizing common path code size are examples of compiler optimizations that limit the amount of accesses to higher Ensemble memory
- The graph below demonstrates how our IRFs (E) consume less energy when compared to an I-Cache(B), fully associative loop cache (A), direct mapped loop cache (D). The loop caches have the same capacity as the IRF.



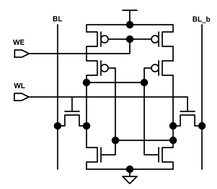

```
@L_BB3:    nop, ld vr1 [ar1+@samples];
           nop, loop.clear pr1 @L_BB3 31;
           nop, ld vr0 [ar0+@coeffs];
@L_BB4:    movi pr2 15, movi pr0 31;
           nop, nop;
@L_BB6:    mov sr0 zr0, nop;
@L_BB8:    mac sr0 vr0 vr1 sr0, loop.clear pr2 @L_BB8 15;
           mac sr0 vr0 vr1 sr0, nop;
@L_BB9:    nop, recv vr1 mr0;
           mov zr0 sr0, loop.clear pr0 @L_BB6 31;
           mov zr0 vr1, send mr3 tr0;
```

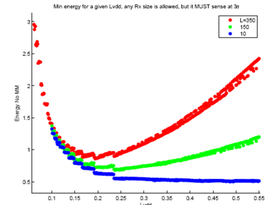
- The compiler schedules code for the distributed, hierarchical memory
- It addresses a phase ordering problem between instruction scheduling and register allocation by scheduling, allocating, then rescheduling instructions
- Using auto-update features of the architecture, the compiler is also able to issue zero overhead loops

Circuits

Augmenting the standard cell flow to further efficiency gains

- In ELM's standard cell design, 25-30% of the energy is lost in wires
- Custom circuits can replace long wires with low swing variants, where signaling is done differentially between 0 and 200mV
- The graph at right shows the energy decrease of a transmitter and receiver for different voltages and wire lengths





- By designing our own memories, we able to make design decisions based on our needs (small, fast arrays)
- By viewing each cell as a small sense amplifier, writes on the cell use a small voltage on the bitlines (3x total savings in a 64x32 array)
- Our 2+2 ported register file, saves about 4x (8x32 entry) the energy per access than a system of flip-flops