Energy Efficient Computing

Kapok & ELM: Reducing the energy cost of parallel computation

**Stanford University**

**ELM: Embedded Computing**

The ELM project focuses on the creation of low-power, high-performance, programmable embedded systems. By designing systems composed of many efficient processor and memory tiles and providing complete programming and efficient run-time environments, ELM will significantly reduce or eliminate the need of fixed-function hardware in many systems.

**Architectural Overview**

- Homogeneous tiles with 4 small processors and 8kB of software controlled memory
- Instructions from software managed instruction register file
- Data register files of varying size
- Hardware engines for data movement

**Hardware Mechanisms For Embedded Software**

<table>
<thead>
<tr>
<th>Software Characteristic</th>
<th>Hardware Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction Level Parallelism</td>
<td>2 Wide VLIW Ensemble Processors</td>
</tr>
<tr>
<td>Regular Data Access Patterns</td>
<td>Hardware Stream Engines for block transfers</td>
</tr>
<tr>
<td>Small Code Size</td>
<td>64-entry, software controlled instruction register file</td>
</tr>
<tr>
<td>Tight Loops</td>
<td>Auto-incrementing loop counters and indexes into both memory and GPR</td>
</tr>
<tr>
<td>Data Level Parallelism</td>
<td>4x-6x in an Ensemble can execute in SIMD mode</td>
</tr>
<tr>
<td>Thread Level Parallelism</td>
<td>Many core shared address space chip-level architecture</td>
</tr>
</tbody>
</table>

**Software Design Flow**

- Developers write high level code as streams, kernels and throughput constraints
- The programming system optimizes and connects kernels
- The compiler optimizes and parallelizes low-level kernels

**Communication**

- Intra-Ensemble communication via zero overhead message registers
- Software/hardware controlled on chip memories stages data transfers
- Hardware stream descriptors and block memory transfer mechanisms provide efficient data movement

**Exposed Data Locality**

- Put data closest to the processor(s) that are currently using it
- Less energy spent locating and moving the data on loads and stores
- Managed either via hardware or software

**Power Efficiency**

- 55% due to power requirements
- 3x-6x more efficient than competitive designs

**Motivation**

<table>
<thead>
<tr>
<th>TCO of a Data Center</th>
<th>Power = ( E_{\text{Operation}} \times \frac{\text{Operations}}{\text{Second}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server Overhead</td>
<td>11%</td>
</tr>
<tr>
<td>Server Processor</td>
<td>8%</td>
</tr>
<tr>
<td>Server Power</td>
<td>28%</td>
</tr>
<tr>
<td>Server Capital</td>
<td>8%</td>
</tr>
<tr>
<td>Data Center</td>
<td>8%</td>
</tr>
<tr>
<td>Data Center Op-Ex</td>
<td>1%</td>
</tr>
<tr>
<td>Memory Power</td>
<td>10%</td>
</tr>
<tr>
<td>Fixtures</td>
<td>10%</td>
</tr>
</tbody>
</table>

**Key Results & Future Work**

- Implemented ELM’s compute and memory tiles
- Demonstrated ELM to be 3-4x more efficient than conventional RISC cores
- Developed a working compiler and programming system, translating C++ to ELM assembly
- In the process of open-sourcing the RTL

**Remote Messaging & Communication**

- Access highly contained variables/locks at their home node via active messages
- Fast barriers for decreasing synchronization overhead
- Configurable cache hierarchy allows programmers to take advantage of different forms of sharing

**Graph Algorithms**

- Highly irregular with little locality

**Memory Access Patterns**

**Application**

- Matrix Operations: Predictable blocks, suited for software
- Particle Simulation: Neighborhoods, needs coalescing scatters/gathers

**Energy per CRC**

<table>
<thead>
<tr>
<th>Energy (AU)</th>
<th>Operations</th>
<th>Server Power</th>
<th>Server Processor</th>
<th>Server Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>1</td>
<td>11%</td>
<td>8%</td>
<td>11%</td>
</tr>
<tr>
<td>0.4</td>
<td>2</td>
<td>11%</td>
<td>8%</td>
<td>11%</td>
</tr>
<tr>
<td>0.6</td>
<td>3</td>
<td>11%</td>
<td>8%</td>
<td>11%</td>
</tr>
</tbody>
</table>

**Graph**

- Estimates of the latency and energy savings of using active messages instead of loads and stores

**Kapok: Supercomputing**

The goal of the Efficient Supercomputing project is to significantly reduce the amount of energy consumed executing scientific code while providing programmers an API that allows for productive algorithm implementation. We do this by exposing locality to the programmer, minimize unnecessary network traffic, and reduce cache contention and meta-data overhead.