Trend towards heterogeneous systems

- Clear that microprocessor trends have shifted since 2005

[https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data]

Limited single thread performance
- Thermal and power budget
- Dark silicon effect

Increase core counts
Specialize cores

Exploit heterogeneity
Micro-architectural heterogeneity is already here

ARM DynamIQ / big.LITTLE

High-performance big cores

Energy-efficient LITTLE cores

Compute capacity (Performance)

Power

iPhone X

Galaxy S8
Micro-architectural heterogeneity is already here

ARM big.LITTLE / DynamIQ

But only for homogeneous instruction set architecture (ISA)

Can we utilize heterogeneous-ISA?
Different ISA, different execution profile

- “Harnessing ISA Diversity: Design of a Heterogeneous-ISA Chip Multiprocessor,” Venkat and Tullsen (UCSD), ISCA’14
  
  - RISC vs CISC
  - Register memory architecture vs load/store architecture
  - Vector instruction support (e.g., SIMD)
  - Power efficiency per instruction
  - Pipeline depth
  - Degree of parallelism
Different ISA, different execution profile

- “Harnessing ISA Diversity: Design of a Heterogeneous-ISA Chip Multiprocessor,” Venkat and Tullsen (UCSD), ISCA’14

Performance of bzip2 for different peak power budgets
ISA affinity opens up opportunities

- Can improve performance and energy consumption by migrating work to an optimal-ISA node

- **Homogeneous**
  - Alpha

- **Single-ISA**
  - big Alpha
  - medium Alpha
  - little Alpha

- **Heterogeneous-ISA**
  - ARM’s thumb
  - x86_64
  - Alpha

---

**Performance**

- Area (mm²): 48, 64, 80, Unlimited
- Speedup: 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0

**EDP**

- Area (mm²): 48, 64, 80, Unlimited
- Normalized EDP: 0.2, 0.4, 0.6, 0.8, 1.0, 1.2
Challenges in exploiting the ISA affinity

• Relocate execution across machine boundaries
  – Single-chip/board heterogeneous-ISA architecture is not available
  – Not obvious even between homogeneous-ISA machines

• Deal with discrepancies between ISAs
  – Let assume ISAs have the same endian and primitive data type size
  – However, register set, stack layout, executable layout, …

• Want to run applications as-is
  – Cost(developer/software) >>>>> cost(hardware)
  – Can enable future-proofing – important for legacy!
Popcorn Linux considers programmability

**Serial**

```c
void full_verify(void)
{
    INT_TYPE i, j;
    for( i=0; i<NUM_KEYS; i++ )
        key_buff2[i] = key_array[i];
    for( i=0; i<NUM_KEYS; i++ )
        key_array[--key_buff_ptr_global[key_buff2[i]]-total_lesser_keys] = key_buff2[i];
}
```

**MPI**

```c
void full_verify(void)
{
    MPI_Status status;
    MPI_Request request;
    INT_TYPE i, j;
    INT_TYPE k, last_local_key;
    for( i=0; i<total_local_keys; i++ )
        key_array[--key_buff_ptr_global[key_buff2[i]]-total_lesser_keys] = key_buff2[i];
    last_local_key = (total_local_keys<1) ? 0 : (total_local_keys-1);
    if( my_rank > 0 )
        MPI_Irecv( &k, 1, MPI_KEY_TYPE, my_rank-1, 1000, MPI_COMM_WORLD, &request );
    if( my_rank < comm_size-1 )
        MPI_Send( &key_array[last_local_key], 1, MPI_KEY_TYPE, my_rank+1, 1000, MPI_COMM_WORLD );
    if( my_rank > 0 )
        MPI_Wait( &request, &status );
    ...
}
```

**OpenCL**

```c
void full_verify( void )
{
    cl_kernel k_fv0, k_fv1;
    cl_mem m_j; cl_int ecode;
    INT_TYPE *g_j;
    INT_TYPE i, j;
    size_t j_size;
    size_t fv0_lws[1], fv0_gws[1];
    size_t fv1_lws[1], fv1_gws[1];
    j_size = sizeof(INT_TYPE) * (FV2_GLOBAL_SIZE / FV2_GROUP_SIZE);
    m_j = clCreateBuffer(context, CL_MEM_READ_WRITE, j_size, NULL, &ecode);
    k_fv1 = clCreateKernel(program, "full_verify1", &ecode);
    k_fv0 = clCreateKernel(program, "full_verify0", &ecode);
    ecode = clSetKernelArg(k_fv0, 0, sizeof(cl_mem), (void*)m_key_array);
    ecode |= clSetKernelArg(k_fv0, 1, sizeof(cl_mem), (void*)m_key_buff2);
    fv0_lws[0] = work_item_sizes[0];
    fv0_gws[0] = NUM_KEYS;
    ecode = clEnqueueNDRangeKernel(cmd_queue, k_fv0, 1, NULL, fv0_gws[0], fv0_lws[0], 0, NULL, NULL);
    ecode = clSetKernelArg(k_fv1, 0, sizeof(cl_mem), (void*)m_key_buff2);
    ecode |= clSetKernelArg(k_fv1, 1, sizeof(cl_mem), (void*)m_key_buff1);
    fv1_lws[0] = work_item_sizes[0];
    fv1_gws[0] = NUM_KEYS;
    ecode = clEnqueueNDRangeKernel(cmd_queue, k_fv1, 1, NULL, fv1_gws[0], fv1_lws[0], 0, NULL, NULL);
    ...
}
```

NPB IS
Popcorn Linux

Software framework to run applications “as-is” on heterogeneous-ISA hardware

http://popcornlinux.org
Outline

- What for heterogeneous-ISA systems?
- Introduction to Popcorn Linux
- Our approaches in Popcorn Linux
  - Compiler
  - Runtime
  - Operating System
- Ongoing work
Previously:

**Popcorn Linux for replicated kernels**

- Run multiple kernels on a single system
  - Run a kernel on a subset of processors in a system
  - Primarily for OS scalability
- Provide a single system image over the multiple kernels
- Migrate processes across the kernel boundary
Popcorn Linux for heterogeneous ISAs

- Extend the replicated kernel concept over multiple nodes
  - Exploit the execution migration feature
- Allow threads in a process to be split over multiple nodes
- Support execution migration across ISA-different nodes
Popcorn Linux yields performance and energy gains over homogenous-ISA

- “Breaking the boundaries in heterogeneous-ISA datacenters,” Barbalace et al., ASPLOS’17
  - Workload sets drawn from HPC benchmark suite (NPB)
  - Yields 30% energy savings on average (max is 66% for set-3)
How Popcorn Linux work?

**Runtime**: Transform dynamic, ISA-specific program states

**Kernel**: Migrate execution and provide a distributed execution environment

**Compiler**: Generate multi-ISA binary

- Application source (.c) → Popcorn compiler toolchain → Popcorn Multi-ISA Binary
Popcorn Compiler

Compilation

• Built on top of clang/LLVM

• Application source lowered into LLVM IR
  – Insert migration points
    • Migration only at “equivalence points”; e.g., function entry/exit
  – Analyze liveness of variables

• IR passed through each ISA backend for generating code
  – Instrumentation to generate metadata (e.g., live locations)

• A post-process aligns code and data in uniform layout
Popcorn Compiler

Multi-ISA binary

- Migratable across ISAs
  - Single `.data` section, multiple `.text` sections (one per-ISA)
  - Global data (.data), code (.text) and TLS aligned across all compilations
    - Pointers are valid across all ISAs
  - State transformation metadata
    - Added to binary for translating registers/stack between ISA-specific formats
**Popcorn Compiler**

**Multi-ISA binary**

- Migratable across ISAs
  - Single `.data` section, multiple `.text` sections (one per-ISA)
  - Global data (.data), code (.text) and TLS aligned across all compilations
  - Pointers are valid across all ISAs
- State transformation metadata
  - Added to binary for translating registers/stack between ISA-specific formats
Popcorn Runtime

- Transform registers and stack between ISA-specific formats
  - Refer to the transformation metadata in the binary

- Two-phase process
  - Read compiler metadata describing function activation layouts
  - Rewrite stack in its entirety from source to destination ISA format

- After transformation, runtime invokes migration
  - Pass destination ISA’s register state and stack to OS
Stack transformation

Function: foo
Call site: 193
Call frame size: 32 bytes
Return address: 0x412820

Function: baz
Call site: 10
Call frame size: 32 bytes
Return address: 0x410548

Function: bar
Call site: 37
Call frame size: 16 bytes
Return address: 0x410204

Function: baz
Call site: 10
Call frame size: 48 bytes
Return address: 0x410532

Function: foo
Call site: 193
Call frame size: 40 bytes
Return address: 0x412700

Function: bar
Call site: 37
Call frame size: 32 bytes
Return address: 0x410198

Source

Destination

Top of Stack
Popcorn Runtime

Stack transformation

Function: foo
Call site: 193
Call frame size: 40 bytes
Return address: 0x412700

Function: bar
Call site: 37
Call frame size: 32 bytes
Return address: 0x410198

foo() call frame
foo()'s registers
foo()'s Return Address
foo()'s Saved FBP

bar() call frame
bar()'s registers
bar()'s Return Address
bar()'s Saved FBP

baz() call frame
baz()'s registers
baz()'s Return Address
baz()'s Saved FBP

Runtime

foo() call frame
foo()'s registers
foo()'s Return Address
foo()'s Saved FBP

bar() call frame
bar()'s registers
bar()'s Return Address
bar()'s Saved FBP

baz() call frame
baz()'s registers
baz()'s Return Address
baz()'s Saved FBP

...'s Return Address
...'s Saved FBP

Function: foo
Call site: 193
Call frame size: 40 bytes
Return address: 0x412700

Function: bar
Call site: 37
Call frame size: 32 bytes
Return address: 0x410198

foo() call frame
foo()'s registers
foo()'s Return Address
foo()'s Saved FBP

bar() call frame
bar()'s registers
bar()'s Return Address
bar()'s Saved FBP

baz() call frame
baz()'s registers
baz()'s Return Address
baz()'s Saved FBP
Popcorn Runtime

Stack transformation

Stack

Register set mapped to target architecture

Invoke migration

Popcorn Kernel
Popcorn Kernel

- Based on Linux kernel v4.4.55
  - Working on x86-64 and aarch64

- Tried to be **architecture-agnostic**
  - Except for register and PTE manipulation

- Relocate/distribute threads over multiple nodes

- Migrating entire memory is infeasible
- Should provide sequential consistency
Migrating execution

• Equivalent to context switching across machines
  
  - At origin: Save the execution context
    - The runtime provides the register set
    - thread_struct + mm_struct
  
  - At remote: Restore the context on a thread
    - Fork a kernel thread, and downgrade it to a user thread
    - Construct mm_struct and associate it with the thread
    - Setup register set and thread_struct
    - Return from the kernel space

  ➔ Resume execution as if returned from system call
Distributed thread execution in action

View from applications

Process A

High-speed low-latency interconnect

Actual execution

Node 0 (Remote)
Remote thread
Single thread migration
Exclusive page access for writes

Node 1 (Origin)
Original threads
Shared page access for reads
Fetch VMA on demand
Multiple thread relocation

Node 2 (Remote)
Remote threads

Writable page
Readable page
Invalid page
VMA
Providing a consistent memory view to distributed threads

• The origin controls the ownership and data
  – Origin owns all pages in the beginning
  – Contact origin to get an ownership and data for pages

• Read-replicate, write-invalidate protocol at page granularity
  – To exploit the common cases in memory-intensive workloads

• Implemented in the virtual memory system in operating system
  – Transparent to the application’s perspective
Taming concurrent page faults with leader/follower model

• Coalesce multiple faults and handle with a single operation

• Leader
  – The first thread that starts a page fault operation for a page at a moment
  – Execute the fault handling operation for the page
    • E.g., bring the page from remotes, fix up page table, flush TLB, …

• Followers
  – Threads that can utilize the leader’s outcome
  – Wait for the completion of the leader’s fault handling

• Otherwise
  – Wait or retry
Reducing false page sharing

• Inherent in page-level consistency protocol
  – A page can bounce between nodes if they access different data object in the same page

• Behavior analysis tool helps to identify false page sharing
  – Analyze page fault events collected in profiling mode
  – Pinpoint to the location in code
    • # of faults, type of faults, type of program objects
Reducing false page sharing

- Inherent in page-level consistency protocol
  - A page can bounce between nodes if they access different data object in the same page

- Behavior analysis tool helps to identify false page sharing
  - Analyze page fault events collected in profiling mode
  - Pinpoint to the location in code
    - # of faults, type of faults, type of program objects

<table>
<thead>
<tr>
<th>Application</th>
<th>Mod. LoC</th>
<th>Application</th>
<th>Mod. LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td></td>
<td>PARSEC</td>
<td></td>
</tr>
<tr>
<td>Grep</td>
<td>+21</td>
<td>Blackscholes</td>
<td>-</td>
</tr>
<tr>
<td>Kmeans</td>
<td>+6</td>
<td>-3</td>
<td></td>
</tr>
<tr>
<td>NPB Common</td>
<td>+1</td>
<td>-1</td>
<td>Polymer Common</td>
</tr>
<tr>
<td>Polymer</td>
<td></td>
<td>+86</td>
<td>-67</td>
</tr>
<tr>
<td>BT</td>
<td>+5</td>
<td>-2</td>
<td>BFS</td>
</tr>
<tr>
<td>EP</td>
<td>+2</td>
<td>-1</td>
<td>BP</td>
</tr>
<tr>
<td>FT</td>
<td>+1</td>
<td>-1</td>
<td>PageRank</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+32</td>
<td>-30</td>
</tr>
</tbody>
</table>

Took 4 days for a Ph.D. student to reduce false page sharing from 9 applications.
The memory consistency protocol allows applications to scale their performance.
Summary: How Popcorn Linux work?

- **Compiler** generates “multi-ISA” binaries
  - `.text` for every ISA (symbol-aligned)
  - One `.data` for the entire machine (symbol-aligned)

- **Runtime** transforms dynamic ISA-specific program state
  - Stack, registers, etc, re-written on the fly

- **Operating system** migrates execution and provides a consistent execution environment across machines
  - Guarantee sequential data consistency to distributed threads
Ongoing work

• Towards a heterogeneous rack-scale system
  – Previously: ARM + x86 prototype platform
  – aarch64
    • APM883208-X1
      – 8 cores @2.4GHz
    • 16GB RAM, PCIe 8x
  – x86_64
    • Intel Xeon E5-1650v2
      – 6 cores 2HT @3.50 GHz
    • 16GB RAM, PCIe 8x
Rack-scale prototype platform

Currently on-line
- 8 Intel Xeon (x86_64)
- 8 Cavium ThunderX (ARM64)

Working on
- APM X-Gene2
- IBM Power8
- RISC-V
Interconnects

• Dolphin interconnect
  – Dolphin PXH810 over PCIe up to 56Gb/s
  – Between tightly coupled nodes

• InfiniBand
  – Mellanox ConnectX-4/3 NICs and SX6036 switch up to 56Gb/s
  – Utilize Remote DMA (RDMA) feature
  – For global communication

• Ethernet
  – Based on the standard TCP/IP and sockets
  – As a standard, versatile interconnect
Ongoing work

• Towards a heterogeneous rack-scale system
• Incorporate more heterogeneity
  – IBM Power8
  – RISC-V
• Task scheduling in a heterogeneous-ISA rack
• Popcorn as a security infrastructure
  – E.g., Increase entropy to prevent ROP attacks
• Cross-ISA execution in a virtualization setting
Thank you!

* Popcorn Linux Team *

**Supervising:** Changwoo Min, Binoy Ravindran  
**Compiler, runtime:** Anthony Carno, Mohamed Karaoui, Robert Lyerly  
**Kernel:** Horen Chuang, Sang-Hoon Kim
Backup slides
Performance and energy gains over homogenous-ISA

- Workload sets drawn from HPC benchmark suite (NPB)
- Smaller the energy-delay product, the better

- Popcorn yields 30% energy savings on average (max is 66% for set-3)
- Popcorn yields 11% reduction in EDP

["Breaking the boundaries in heterogeneous-ISA datacenters," Barbalace et al., ASPLOS'17]
ISA affinity opens up opportunities

- “The Impact of ISAs on Performance,” Akram and Sawalha, WDDD/ISCA’17
- “OS Support for Thread Migration and Distribution in the Fully Heterogeneous Datacenter,” Olivier et al., HotOS’17

PARSEC blackscholes
Motivation

• Proliferation of heterogeneous-ISA platforms
  – Discrete
    • Xeon Phi, GPUs
  – Integrated On-Die/SoC
    • CPU + GPU (AMD A-series)
    • CPU + Accelerator Slices (Tilera TILEncore Gx-series)
    • CPU + GPU + DSP + … (Qualcomm Snapdragon)

• Mix of OS & non-OS capable
Instruction Set Architecture (ISA)

Reduced Instruction Set Computer (RISC)

Complex Instruction Set Computer (CISC)

ARM Compiler
Language Parser
Optimization
Binary Emitter

x86 Compiler
Language Parser
Optimization
Binary Emitter

ARM Instruction Set Overview

Intel 64 and IA-32 Architectures
Software Developer’s Manual
Volume 2A, 2B, 2C, 2D, 2E, 2F
Instruction Set Reference, A.2
Popcorn compilation

- Built on top of clang/LLVM
  - clang/LLVM 3.7.1, GNU gold 2.27 (~12.4k LoC)
  - Address space alignment (~700 LoC), post-processing (~1.7k LoC) tools
  - State transformation/migration libraries (~5.9k LoC)
  - Minor updates to musl-libc 1.1.18, libelf, and GNU OpenMP runtime
State Transformation

• How fast is state transformation?
  – Reference: typically scheduling is done at every 10 milliseconds
Migration Points

Figure 3. NPB CG number of instructions between migration points.

Figure 4. NPB IS number of instructions between migration points.

Figure 5. NPB FT number of instructions between migration points.
**Significant rewriting cost:**

**NPB example**

- **From **shared memory/OpenMP** to MPI**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>CG</th>
<th>EP</th>
<th>FT</th>
<th>IS</th>
<th>MG</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenMP LOC</td>
<td>1150</td>
<td>297</td>
<td>1106</td>
<td>1108</td>
<td>1481</td>
</tr>
<tr>
<td>MPI modified</td>
<td>98%</td>
<td>44%</td>
<td>98%</td>
<td>46%</td>
<td>97%</td>
</tr>
</tbody>
</table>

OpenMP and MPI version of NASA NPB

- **From **serial code** to OpenCL**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>CG</th>
<th>EP</th>
<th>FT</th>
<th>IS</th>
<th>MG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial LOC</td>
<td>506</td>
<td>163</td>
<td>606</td>
<td>454</td>
<td>852</td>
</tr>
<tr>
<td>OpenCL added</td>
<td>303</td>
<td>164</td>
<td>143</td>
<td>177</td>
<td>189%</td>
</tr>
</tbody>
</table>

OpenCL and serial version of SNU NPB

“Popcorn: bridging the programmability gap in heterogeneous-ISA platforms,” A. Barbalace et al., EuroSys, 2015.
Thread migration in action

Migrate to ARM

Original thread
send page containing
sp + 0xbeef

Kernel 1

Page fault at
sp + 0xbeef

Remote thread

Kernel 2

Page fault at @str

str x1, [sp,#0xbeef]
Popcorn Kernel

Migrate threads

Provide consistent memory
The Rack

• Bundle
  – The building block for “The Rack”
  – A set of nodes that are tightly-coupled each other
    • To control the latency of memory consistency protocol

• Bundles are connected via a high-speed switching interconnect