Architecture Support for Customization and Specialization

Jason Cong
UCLA Computer Science Department
cong@cs.ucla.edu
The Power Barrier and Current Solution

- 10’s to 100’s cores in a processor
- 1000’s to 10,000’s servers in a data center

Source: Shekhar Borkar, Intel
Cost and Energy are Still a Big Issue …

Hiding in Plain Sight, Google Seeks More Power

Google is building two computing centers, top and left, each the size of a football field, in The Dalles, Ore.

By JOHN MARKOFF and SAUL HANSELL
Published: June 14, 2008

THE DALLES, Ore., June 8 — On the banks of the windswept Columbia River, Google is working on a secret weapon in its quest to dominate the next generation of Internet computing. But it is hard to keep a secret when it is a computing center as big as two football fields, with twin cooling plants protruding four stories into the sky.

Cost of computing
• HW acquisition
• Energy bill
• Heat removal
• Space
• …
Next Significant Opportunity -- Customization

Parallelization
Customization
Adapt the architecture to Application domain
### Justification 1 – Potential of Customization

<table>
<thead>
<tr>
<th>AES 128bit key</th>
<th>Throughput</th>
<th>Power</th>
<th>Figure of Merit (Gb/s/W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>128bit data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.18µm CMOS</td>
<td>3.84 Gbits/sec</td>
<td>350 mW</td>
<td>11 (1/1)</td>
</tr>
<tr>
<td>FPGA [1]</td>
<td>1.32 Gbit/sec</td>
<td>490 mW</td>
<td>2.7 (1/4)</td>
</tr>
<tr>
<td>ASM StrongARM [2]</td>
<td>31 Mbit/sec</td>
<td>240 mW</td>
<td>0.13 (1/85)</td>
</tr>
<tr>
<td>Asm Pentium III [3]</td>
<td>648 Mbits/sec</td>
<td>41.4 W</td>
<td>0.015 (1/800)</td>
</tr>
<tr>
<td>C Emb. Sparc [4]</td>
<td>133 Kbits/sec</td>
<td>120 mW</td>
<td>0.0011 (1/10,000)</td>
</tr>
<tr>
<td>Java [5] Emb. Sparc</td>
<td>450 bits/sec</td>
<td>120 mW</td>
<td>0.0000037 (1/3,000,000)</td>
</tr>
</tbody>
</table>

**Figure of Merit**

1. **AES 128bit key**
   - 128bit data
   - Throughput: 3.84 Gbits/sec
   - Power: 350 mW
   - Figure of Merit (Gb/s/W): 11 (1/1)

2. **FPGA [1]**
   - Throughput: 1.32 Gbit/sec
   - Power: 490 mW
   - Figure of Merit (Gb/s/W): 2.7 (1/4)

3. **ASM StrongARM [2]**
   - Throughput: 31 Mbit/sec
   - Power: 240 mW
   - Figure of Merit (Gb/s/W): 0.13 (1/85)

4. **Asm Pentium III [3]**
   - Throughput: 648 Mbits/sec
   - Power: 41.4 W
   - Figure of Merit (Gb/s/W): 0.015 (1/800)

5. **C Emb. Sparc [4]**
   - Throughput: 133 Kbits/sec
   - Power: 120 mW
   - Figure of Merit (Gb/s/W): 0.0011 (1/10,000)

   - Throughput: 450 bits/sec
   - Power: 120 mW
   - Figure of Merit (Gb/s/W): 0.0000037 (1/3,000,000)

---

[1] Amphion CS5230 on Virtex2 + Xilinx Virtex2 Power Estimator
[4] gcc, 1 mW/MHz @ 120 Mhz Sparc – assumes 0.25 µ CMOS
[5] Java on KVM (Sun J2ME, non-JIT) on 1 mW/MHz @ 120 MHz Sparc – assumes 0.25 µ CMOS

Justification 2 -- Advance of Civilization

- For human brain, Moore’s Law scaling has long stopped
  - The number of neurons and their firing speed did not change significantly
- Remarkable advancement of civilization via specialization
- More advanced societies have higher degree of specialization
Architecture Support for Customization

Key questions: Optimal trade-off between efficiency & customizability
Which options to fix at CHP creation? Which to be set by CHP mapper?

Cache parameters
- Cache size & configuration
- Cache vs SPM
- ...

NoC parameters
- Interconnect topology
- # of virtual channels
- Routing policy
- Link bandwidth
- Router pipeline depth
- Number of RF-I enabled routers
- RF-I channel and bandwidth allocation
- ...

Core parameters
- Frequency & voltage
- Datapath bit width
- Instruction window size
- Issue width
- Cache size & configuration
- Register file organization
- # of thread contexts
- ...

Custom instructions & accelerators
- Shared vs. private accelerators
- Choice of accelerators
- Custom instruction selection
- Amount of programmable fabric
- ...

Customizable Heterogeneous Platform (CHP)
Core spilling – [Cong et al Trans. on Parallel and Distributed Systems 2007]

- CMP systems focus on improving overall throughput
  - Sequential or legacy applications might not see benefits
- Key idea – allow execution to be spilt from one core to next at run-time
  - Simulate increase in register file, instruction queue, ROB and LSQ size
  - Allocate cores intelligently to spilling core

Core A sends 62 compiler visible registers, a 24-entry store buffer, the LSQ entries of any stores that are in-flight, and a PC.

Core B begins fetching from the PC sent by Core A. Core A continues execution, and sends any register or store values that were in-flight at the time of the spill as they complete.

Spilling can continue if B’s resources are exhausted.

Eventually, all instructions on Core A will commit (unless there is an exception or branch misprediction) and Core A can be released into the pool of idle cores.
Core spilling – [Cong et al Trans. on Parallel and Distributed Systems 2007]

- **Results**
  - Core spilling achieves more than 50% of the performance of ‘ideal’ 32-issue core by using 4-issue cores for single applications
  - 39% improvement for multiple application workload
  - Up to 40% reduction in latency for changing workloads
Key questions: Optimal trade-off between efficiency & customizability
Which options to fix at CHP creation? Which to be set by CHP mapper?
Extensive Use of Accelerators [Cong, et al SAW’2011]

- Many-core may not be viable
- Utilization wall
  - 6.5% utilization for a 45 nm chip filled with 64bit operators, assuming a power budget of 80 W [ASPLOS’2010]
- Proposed solution: extensive use of accelerators (customized or implemented using programmable fabric)
  - Better performance
  - Higher power-efficiency
  - It’s ok to be “wasteful”
- Type of accelerators:
  - Tightly vs. loosely coupled
- Needs:
  - Efficient accelerator management
Managing accelerator by OS is expensive (200K+ cycles on Solaris)

In an accelerator rich CMP, management should be cheaper both in terms of time and energy
Overall Architecture of AXR-CMP

- **Architecture of AXR-CMP:**
  - Multiple cores and accelerators
  - Global Accelerator Manager (GAM)
  - Shared L2 cache banks and NoC routers between multiple accelerators
Overall Communication Scheme in AXR-CMP

1. The core requests and the GAM responds with a list (lcacc-req).
2. The core reserves (lcacc-rsv) and waits.
3. The core creates and fills a task buffer and starts the accelerator (lcacc-cmd), the accelerator reads this task buffer, and begins working.
4. When the accelerator finishes its current task it notifies the core. The core then sends a message to the GAM freeing the accelerator (lcacc-free).
**Light-weight Interrupt Support**

- **To reduce OS interrupt service**
  - No need to save context

- **Two main components added:**
  - A table to store ISR info
  - An interrupt controller to queue and prioritize incoming interrupt packets

- **Each thread registers:**
  - Address of the ISR and its arguments
  - lw-int source

- **Sources of lw-int:**
  - GAM responses
    - Accelerator ready
    - Wait time for accelerator
  - Accelerator TLB miss
  - Accelerator task buffer empty

- **Limitations:**
  - Only can be used when running the same thread which LW interrupt belongs to
  - OS-handled interrupt otherwise

---

**Operation** | **Latency (# Cycles)**
--- | ---
1 core | 2 cores | 4 cores | 8 cores | 16 cores
**Interrupt** | 16383 | 20361 | 24022 | 26572 | 28574

---

**lw-reg x y z** | Register service routine y to service interrupts arriving from accelerator x. LWI message packet will be written to z

**lw-jump** | Jump to the service routine for the next pending interrupt. Does nothing if no interrupt is pending.

**lw-ret** | Return from an interrupt service routine.
Accelerator Chaining and Composition

- **Chaining**
  - To have an efficient accelerator to accelerator communication

- **Composition**
  - To create the virtual feeling of having larger accelerators for the applications
Hardware Overhead Analysis

- AutoPilot to synthesize Global Accelerator Manager (GAM) module and DMA-C
  - Area is less than 0.01% of the chip (1cm X 1cm) using 65 nm technology.

<table>
<thead>
<tr>
<th>Module</th>
<th>Clock speed</th>
<th>Area (u^2m)</th>
<th>Power (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAM</td>
<td>2 ns</td>
<td>12270</td>
<td>2.64</td>
</tr>
<tr>
<td>DMA-C</td>
<td>2 ns</td>
<td>10071</td>
<td>0.09</td>
</tr>
</tbody>
</table>
Experimental Results – Performance

- **Performance improvement over SW only approaches:** on average 97X, up to 208X

- **Performance improvement over OS based approaches:** on average 4.1 X, up to 10X
Experimental Results – Energy

Energy improvement vs SW-Only

Energy improvement over SW only approaches: on average 112X, up to 187X

Energy improvement vs LCAcc+OS

Energy improvement over OS based approaches: on average 1.8 X, up to 6.1X
Experimental Results – Increasing Request for Accelerator

Maintain speedup compare to OS-based approach when increasing the number of requests for accelerators
Experimental results – Increasing Number of Accelerators and Data Size

• Step-shape response when increasing the number of accelerators
• Increasing speedup when increasing data size
• Except for the FFT, since communication overhead is high (>90%)

For 4 cores
Impact of Chaining and Light-Weight Interrupt

* Increasing benefits from chaining when increasing the input data size

* lw-int benefit when increasing the number of accelerators
Experimental Results – Estimation Errors

* Core estimation error when increasing the data size is flat except for FFT/3D, because of more TLB misses

* Decreasing GAM estimation error because of the canceling effect
Architecture Support for Customization

Cache parameters
- Cache size & configuration
- Cache vs SPM
- ...

NoC parameters
- Interconnect topology
- # of virtual channels
- Routing policy
- Link bandwidth
- Router pipeline depth
- Number of RF-I enabled routers
- RF-I channel and bandwidth allocation
- ...

Core parameters
- Frequency & voltage
- Datapath bit width
- Instruction window size
- Issue width
- Cache size & configuration
- Register file organization
- # of thread contexts
- ...

Custom instructions & accelerators
- Shared vs. private accelerators
- Choice of accelerators
- Custom instruction selection
- Amount of programmable fabric
- ...

Customizable Heterogeneous Platform (CHP)

Fixed Core

Custom Core

Prog Fabric

Reconfigurable RF-I bus
Reconfigurable optical bus
Transceiver/receiver
Optical interface

Key questions: Optimal trade-off between efficiency & customizability
Which options to fix at CHP creation? Which to be set by CHP mapper?
Customizable Hybrid Cache [ISLPED’2011]

- **Cache**: Hardware-controlled
  - Transparent to software: a fast local copy of the global memory address space

- **Scratchpad Memory (SPM)**: Software-controlled (first proposed by [Panda et al. DATE’97])
  - Not transparent to software: a separate address space from the global address space

- **Cache in conjunction with SPM** (E.g. IBM Cell: Cache for PPE, SPM for SPE)

- **Hybrid Cache**: Flexibly size the cache and SPM based on the application requirements
  - Reconfigurable caches (E.g. Reconfigurable cache for media apps [Ranganathan, ISCA’00], Nvidia Fermi)
  - Cache-locking functions (E.g. Way Stealing [Kluter, et.al., DAC’09])
Cache Set Balancing in Hybrid Cache

- Problem:
  - Partition cache and SPM without adaptation to the cache behavior
  - High demand cache set at runtime: high conflict miss rate

- Previous techniques to balance cache set utilization
  - Techniques requiring serial tag-data array access: not suitable for hybrid cache (typically a primary cache for fast access to the SPM)
    - V-way cache [Qureshi, et.al, ISCA’04]
    - Indirect index cache [Givagis, et.al., DAC’03]
    - Set balancing cache [Rolan, et.al., MICRO’09]
  - Techniques not requiring serial tag-data array access: not energy-efficient
    - Victim cache [Jouppi, ISCA’90]: fully-associative victim cache
    - Balanced cache [Zhang, ISCA’06]: CAM in the cache decoder

Non-uniformed cache sets utilization in hybrid cache
AH-Cache

- **Motivation:** balancing the cache set utilization in hybrid cache in such a way that:
  - Energy-efficient
  - Do not require serial tag/data array access

- **Key idea**
  - Dynamically remap SPM blocks from high-demand cache sets to low-demand cache sets
  - Clean interface to software

- **Challenges**
  - Circular bouncing effect
  - Fast SPM block location and access

- **Proposed Adaptive Hybrid Cache (AH-Cache)**
  - To solve the 1st challenge:
    • An adaptive mapping scheme based on a floating-block-holders queue
  - To solve the 2nd challenge:
    • The look-up operation of the SPM location is hidden in the execution (EX) pipeline of the processor, and a clean software interface is provided as a non-adaptive hybrid cache
SPM Mapping Look-up

**SPM access in non-adaptive hybrid cache**
- The high order bits of the virtual address are checked after computed by ALU to determine whether it is targeting the SPM.
- Imposes a constraint that the SPM base address should be aligned with all the low-order bits as 0.

**SPM Lookup and access in AH-Cache**
- SPM Mapping Lookup Table (SMLT)
  - Max SPM size
  - Set index / Way index / Valid bit
- If performing SPM Lookup and access after the address generation, critical path will be increased.
- Hide SMLT in EX pipeline with address generation
  - Base address: address checking (comparator)
  - Address offset: SMLT lookup
- Constraint on the compiler: The memory reference instructions to the SPM array should be:
  - Base address: the base address of the SPM array
  - Offset: the offset related to the SPM array base address
Cache Set Demand Assessment

**Definitions:**
- *high-demand sets*: cache sets which highly utilize most or all cache blocks
- *low-demand sets*: cache sets that underutilize the available blocks of their set

**Objective:** low-demand sets accommodates proportionally more SPM blocks than the high-demand sets

**Miss rate is not a good metric for cache set demand**
- Streaming access or cache thrashing

**Proposed solution: Victim tag buffer**
- Similar to Miss tag in [Zhang & Asanovic, ISLPED’02], but WITHOUT additional memory overhead
- Reuse the tag array, i.e., an element in tag array is:
  - Tag: if the corresponding cache block is a regular cache block
  - Victim tag: if the corresponding cache block is a SPM block
- At replacement, the tag of the victim block is written into the corresponding set of the VTB
- VTB is accessed at a miss of the regular cache part
  - At a hit in VTB, the set’s VTB counter will be increased by 1
Adaptive Mapping

- **Floating blocks**: cache blocks used to adaptively satisfy the high-demand cache sets
- **Naive scheme**: Make a cache set with a high VTB counter get a floating block from a set with a low VTB counter
  - A low VTB counter only means that this set does not need more floating blocks. It does not mean that it can afford to lose one
  - Circular bouncing effect
- **Floating block holder (FBH) queue**: records the cache sets which currently hold the floating blocks
  - A re-insertion bit in each queue element:
    - Indicates whether this cache set is re-inserted to the queue in the current adaptation interval
  - Each cache set holds a 1-bit insertion flag to indicate whether it has been inserted in the queue in the current interval
  - At the beginning of each interval, all the re-insertion bits in the queue and the insertion flags in the cache sets are reset to 0
Adaptive Mapping (cont’d)

◆ Mapping Scheme:
  ▪ When a cache set A’s VTB counter achieves a threshold $T$, it will check, starting from the head of the FBH queue, until it finds a node with a re-insertion bit of 0., say cache set B
  ▪ Set B will accommodate one SPM block from set A
  ▪ Node B is removed and node A is inserted in FBH queue, re-insertion bit as the current insertion flag of A
  ▪ Insertion flag of set A is set to 1
  ▪ Once all the re-insertion bits in the queue are 1, the remapping of this interval is stopped

◆ With the re-insertion bit as 1, a high-demand set will not give up its floating blocks once it is re-inserted to the queue in a particular interval
  ▪ No circular bouncing effect

◆ Parallel FBH queue search
  ▪ To make the worst-case search latency not affect the L1 miss latency
  ▪ Use a priority decoder to search the re-insertion bits in a separate re-insertion bit table (RIBT)
Evaluation Methodology

- **Performance evaluation**: Simics+GEMS
- **Energy evaluation**: Cacti and McPAT
- **Benchmarks**:
  - MiBench
  - SPEC
  - Medical imaging

**Reference design points**: (All the designs use the same compiler optimization for SPM usage)
- Design N: Non-adaptive hybrid cache, baseline
- Design B: Non-adaptive hybrid cache + Balanced cache [Zhang, ISCA’06]
- Design V: Non-adaptive hybrid cache + Victim cache [Jouppi, ISCA’90]
- Design R: Phase-reconfigurable hybrid cache [Zhang, et.al., ISLPED’02]
- Design A: AH-Cache
- Design S: Static optimized hybrid cache

<table>
<thead>
<tr>
<th>Core</th>
<th>Sun UltraSPARC-III Cu processor core</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>L1 Instruction Cache</strong></td>
<td>16KB/32KB, 2-way set-associative, 64-byte block, 2-cycle access latency, pseudo-LRU</td>
</tr>
<tr>
<td><strong>L1 Data Cache</strong></td>
<td>16KB/32KB, 2-way set-associative, 64-byte block, 2-cycle access latency, pseudo-LRU</td>
</tr>
<tr>
<td><strong>L2 Cache</strong></td>
<td>512KB, 8-way set-associative, 64-byte block, 20-cycle access latency, pseudo-LRU</td>
</tr>
<tr>
<td><strong>Main Memory</strong></td>
<td>4GB, 320-cycle access latency</td>
</tr>
</tbody>
</table>

#Memory references of the evaluated benchmarks

<table>
<thead>
<tr>
<th>jpeg</th>
<th>gsm</th>
<th>susan</th>
<th>hammr</th>
<th>soplex</th>
<th>h264ref</th>
<th>dijkstra</th>
<th>patricia</th>
</tr>
</thead>
<tbody>
<tr>
<td>19.8M</td>
<td>65.1M</td>
<td>76.1M</td>
<td>75.7M</td>
<td>22.1M</td>
<td>196.6M</td>
<td>47.7M</td>
<td>16.8M</td>
</tr>
<tr>
<td>astar</td>
<td>gobmk</td>
<td>biHarmonic</td>
<td>mutualInfo</td>
<td>ricianDenoise</td>
<td>regionGrowing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>93M</td>
<td>256.7M</td>
<td>48.6M</td>
<td>98.2M</td>
<td>7.9M</td>
<td>128.3M</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Performance

- AH-Cache can reduce the L1 data cache (the hybrid cache) miss by 52% compared to baseline, and outperforms design B, V and R by 19%, 22% and 33%, respectively.
- AH-Cache reduces the run time by 18% compared to baseline, and outperforms design B, V and R by 3%, 4% and 12%, respectively.
Energy

- The average number of SPM block migrations of AH-Cache at each interval is 4.4, which results in a runtime / energy overhead of less than 0.1%.
- With the additional energy of the SMLT, VTB, adaptive mapping unit, the proposed AH-Cache can still achieve an energy reduction of 16%, 22% and 7% compared to design B, V and R, respectively.
- AH-Cache achieves energy-runtime-production reductions of 19%, 25% and 18% over the design B, V and R, respectively.

TABLE IV. AVERAGE #SPM BLOCK MIGRATIONS IN EACH 1 MILLION CYCLE INTERVAL (UPPER: 16KB, LOWER: 32KB)

<table>
<thead>
<tr>
<th></th>
<th>jpeg</th>
<th>gsm</th>
<th>susan</th>
<th>hmmr</th>
<th>soplex</th>
<th>h264ref</th>
<th>dijkstra</th>
<th>patricia</th>
</tr>
</thead>
<tbody>
<tr>
<td>16KB</td>
<td>5.68</td>
<td>0.04</td>
<td>1.14</td>
<td>8.66</td>
<td>15.9</td>
<td>20.2</td>
<td>6.39</td>
<td>0.79</td>
</tr>
<tr>
<td></td>
<td>0.28</td>
<td>0.01</td>
<td>1.20</td>
<td>0.45</td>
<td>5.26</td>
<td>4.26</td>
<td>0.62</td>
<td>0.15</td>
</tr>
<tr>
<td>32KB</td>
<td>10.5</td>
<td>4.87</td>
<td>0.03</td>
<td>1.95</td>
<td>0.03</td>
<td>0.03</td>
<td>0.04</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>10.3</td>
<td>2.65</td>
<td>0.03</td>
<td>0.03</td>
<td>0.03</td>
<td>0.03</td>
<td>0.04</td>
<td>0.01</td>
</tr>
</tbody>
</table>
Architecture Support for Customization

Key questions: Optimal trade-off between efficiency & customizability
Which options to fix at CHP creation? Which to be set by CHP mapper?

Cache parameters
- Cache size & configuration
- Cache vs SPM
- ...

Core parameters
- Frequency & voltage
- Datapath bit width
- Instruction window size
- Issue width
- Cache size & configuration
- Register file organization
- # of thread contexts
- ...

NoC parameters
- Interconnect topology
- # of virtual channels
- Routing policy
- Link bandwidth
- Router pipeline depth
- Number of RF-I enabled routers
- RF-I channel and bandwidth allocation
- ...

Custom instructions & accelerators
- Shared vs. private accelerators
- Choice of accelerators
- Custom instruction selection
- Amount of programmable fabric
- ...

Customizable Heterogeneous Platform (CHP)

Fixed Core
Fixed Core
Fixed Core
Fixed Core

Custom Core
Custom Core
Custom Core
Custom Core

Prog Fabric
Prog Fabric
 accelerators
 accelerators

Reconfigurable RF-I bus
Reconfigurable optical bus
Transceiver/receiver
Optical interface
Multiband RF-Interconnect

- In TX, each mixer up-converts individual baseband streams into specific frequency band (or channel)
- N different data streams (N=6 in exemplary figure above) may transmit simultaneously on the shared transmission medium to achieve higher aggregate data rates
- In RX, individual signals are down-converted by mixer, and recovered after low-pass filter
Terahertz VCO in 65nm CMOS

Demonstrated an ultra high frequency and low power oscillator structure in CMOS by adding a negative resistance parallel tank, with the fundamental frequency at 217GHz and 16.8 mW DC power consumption.

The measured 4th and 6th harmonics are about 870GHz and 1.3THz, respectively.

Measured signal spectrum with uncalibrated power

higher harmonics (4th and 6th harmonics) may be substantially underestimated due to excessive water and oxygen absorption and setup losses at these frequencies.

“Generating Terahertz Signals in 65nm CMOS with Negative-Resistance Resonator Boosting and Selective Harmonic Suppression” Symposium on VLSI Technology and Circuits, June 2010
Mesh Overlaid with RF-I [HPCA ’08]

- 10x10 mesh of pipelined routers
  - NoC runs at 2GHz
  - XY routing
- 64 4GHz 3-wide processor cores
  - Labeled aqua
  - 8KB L1 Data Cache
  - 8KB L1 Instruction Cache
- 32 L2 Cache Banks
  - Labeled pink
  - 256KB each
  - Organized as shared NUCA cache
- 4 Main Memory Interfaces
  - Labeled green
- RF-I transmission line bundle
  - Black thick line spanning mesh
**RF-I Logical Organization**

- Logically:
  - RF-I behaves as set of N express channels
  - Each channel assigned to src, dest router pair \((s,d)\)

- Reconfigured by:
  - remapping shortcuts to match needs of different applications
**Related Research in CDSC (Center for Domain-Specific Computing)**

**Customizable Heterogeneous Platform**

- **Fixed Core**
- **Custom Core**
- **Prog Fabric**
- **Reconfigurable RF-I bus**
- **Reconfigurable optical bus**

**Domain-specific-modeling** (healthcare applications)

- **Application modeling**
- **Architecture modeling**
- **Domain characterization**

**CHP creation**
- Customizable computing engines
- Customizable interconnects

**Design once**

**CHP mapping**
- Source-to-source CHP mapper
- Reconfiguring & optimizing backend
- Adaptive runtime

**Invoke many times**
Center for Domain-Specific Computing (CDSC)

Aberle (UCLA)
Baraniuk (Rice)
Bui (UCLA)
Chang (UCLA)
Cheng (UCSB)
Cong (Director) (UCLA)
Palsberg (UCLA)
Potkonjak (UCLA)
Reinman (UCLA)
Sadayappan (Ohio-State)
Sarkar (Associate Dir) (Rice)
Vese (UCLA)
CHP Mapping – Compilation and Runtime Software Systems for Customization

Goals: Efficient mapping of domain-specific specification to customizable hardware

– Adapt the CHP to a given application for drastic performance/power efficiency improvement

Domain-specific applications

Programmer

Domain-specific programming model
(Domain-specific coordination graph and domain-specific language extensions)

Source-to source CHP Mapper

C/C++ code

Analysis annotations

C/SystemC behavioral spec

C/C++ front-end

RTL synthesizer (xPilot)

Reconfiguring and optimizing back-end

Binary code for fixed & customized cores

Customized target code

RTL for programmable fabric

Adaptive runtime
Lightweight threads and adaptive configuration

CHP architectural prototypes
(CHP hardware testbeds, CHP simulation testbed, full CHP)

Performance feedback

Abstract execution

Application characteristics

CHP architecture models
**xPilot: Behavioral-to-RTL Synthesis Flow [SOCC’2006]**

- **Behavioral spec. in C/C++/SystemC**
- **Platform description**
- **Frontend compiler**
- **SSDM**
- **RTL + constraints**
- **FPGAs/ASICs**

**Advanced transformation/optimizations**
- Loop unrolling/shifting/pipelining
- Strength reduction / Tree height reduction
- Bitwidth analysis
- Memory analysis …

**Core behavior synthesis optimizations**
- Scheduling
- Resource binding, e.g., functional unit binding register/port binding

**µArch-generation & RTL/constraints generation**
- Verilog/VHDL/SystemC
- FPGAs: Altera, Xilinx
- ASICs: Magma, Synopsys, …
AutoPilot Compilation Tool (based UCLA xPilot system)

- Platform-based C to FPGA synthesis
- Synthesize pure ANSI-C and C++, GCC-compatible compilation flow
- Full support of IEEE-754 floating point data types & operations
- Efficiently handle bit-accurate fixed-point arithmetic
- More than 10X design productivity gain
- High quality-of-results

Developed by AutoESL, acquired by Xilinx in Jan. 2011
Example: Acceleration of Lithographic Simulation [FPGA’08]

- Lithography simulation
  - Simulate the optical imaging process
  - Computational intensive; very slow for full-chip simulation

- XtremeData X1000 development system (AMD Opteron + Altera StratixII EP2S180)

- 15X+ Performance Improvement vs. AMD Opteron 2.2GHz Processor
- Close to 100X improvement on energy efficiency
  - 15W in FPGA comparing with 86W in Opteron

\[ I(x,y) = \sum \lambda_\kappa \ast \left( \sum \tau [\psi_\kappa(x-x_1, y-y_1) - \psi_\kappa(x-x_2, y-y_1) + \psi_\kappa(x-x_1, y-y_2) - \psi_\kappa(x-x_1, y-y_2)] \right)^2 \]
Three SPM management schemes for kernel of 401.bzip (SPEC’06)

for $i = 4$ to $100$

eclass += fmap[i] + fmap[i - 4];

<table>
<thead>
<tr>
<th></th>
<th>Prefetch-only</th>
<th>Reuse-only</th>
<th>Reuse-aware prefetch</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPM size</td>
<td>42</td>
<td>4</td>
<td>25</td>
</tr>
<tr>
<td>Latency hiding</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

prefetch latency $P = 20$

reuse distance $d = 4$
A Reuse-Aware SPM Prefetching Algorithm

Contributions

- Propose RASP (Reuse-Aware SPM Prefetching) scheme
  - Hide memory access latency
  - Reduce number of data transfers
- Quantify the impact of regular reuse pattern on SPM prefetching decisions
A Reuse-Aware SPM Prefetching Algorithm

- Reuse candidate graph construction
  - Each vertex represents one array reference
  - Each edge represents reuse dependency

\[
\text{for } i = 0 \text{ to } N \\
\text{for } j = 0 \text{ to } M \\
v[i][j] = u[i+1][j+1] + \text{DT} \ast (u[i+1][j] + u[i][j+1] + u[i][j] + u[i+1][j+2] + u[i+2][j+1])
\]
A Reuse-Aware SPM Prefetching Algorithm

Problem Statement

- Given a set of reuse candidate graphs constructed for a loop nest and maximal SPM size
- Create SPM buffer for each vertex to hide memory access latency as well as minimizing the number of data transfers between SPM and conventional memory
A Reuse-Aware SPM Prefetching Algorithm

Prefetch latency: 20
SPM\text{max}: 3KB

< 3KB SPM, reduce ~60% data transfers!
A Reuse-Aware SPM Prefetching Algorithm

- **Performance & energy comparison**

  - **Average performance gain** ~ 15.9%, 12.9% and 18.5%
  - **Average energy gain** ~ 22%, 31.2% and 10%
We believe that customization is the next transformative approach to energy efficient computing.

Many opportunities and challenges for architecture support:
- Cores
- Accelerators
- Memory
- Network-on-chips

Software support is also critical (more related research in CDSC)
Acknowledgements

- This research is partially supported by GSRC and CDSC supported by NSF
- Collaborators

Jason Cong  Glenn Reinman  Frank Chang

Mohammad Ali Ghodrat

Michael Gill  Karthik Gururaj  Hui Huang  Chunyue Liu  Yi Zou