Architectural Adaptation for Application-Specific Locality Optimizations

Xingbin Zhang  Ali Dasdan Martin Schulz
Rajesh K.Gupta  Andrew A.Chien

Department of Computer Science
University of Illinois at Urbana-Champaign

Motivation

- Projections by the SIA show that on-chip system performance will increasingly dominated by interconnect delays.
  - Due to interconnect delays, the on-chip clock periods will be limited to 1 nanosecond.
  - The unit gate delay scales down to 20 pico-seconds.
  - The decreasing delay penalty for programmable logic blocks compared to interconnect delays makes incorporation of small programmable logic attractive.

Motivation

- Advances in circuit modeling using HDLs such as Verilog and VHDL.
  - The process of hardware design is increasingly a language-level activity, supported by compilation and synthesis tools.

Motivation

- Current architecture to exploit programmable logic is on eof co-processing — a processor working in conjunction with dedicated hardware assists to deliver a specific application.
Motivation

• Drawback of co-processing approach:
  – A system generated using this approach typically cannot be retargeted to another application without repartitioning hardware and software functionality and reimplementing co-processing hardware.
  – It presents an obstacle of exploiting programmable logic for general-purpose computing even though technology trends made it possible.

Architectural Adaptation

• Integrate small blocks of programmable logic into key elements of a baseline architecture.

Architectural Adaptation

• Customize architectural mechanisms and policies to match an application.
• It can be used in the binding, mechanisms, and policies on the interaction of processing, memory, and communication resources while keeping the macro-level organization the same.
• The adaptation can be done statically or at runtime.

Architectural Adaptation

• Advantage
  – Provides the mechanisms for application-specific hardware assists to overcome the rigid architectural choices in modern computer systems that do not work well across different applications.
  – The integration of programmable logic with memory components enables application-specific locality optimizations, which can overcome long latency and limited bandwidth in the memory hierarchy.
  – Preserve the machine usability through software.
Architectural Adaptation

• Disadvantage
  – The potential increase on system design and verification time due to the addition of programmable logic.

Case Studies

• Background
  – On modern architectures with deep memory hierarchy, data transfer bandwidth and access latency differentials across levels of magnitude, making locality optimization critical for performance.
  – Compiler optimizations can be effective for regular applications, and optimizations for irregular applications can greatly benefit from architectural support. But no fixed architectural policies or mechanisms work well for all applications.

• Data Structure
  – For our application examples, we use the sparse matrix library SPARSE, the data structure for sparse matrix is as follows:

```c
struct MatrixElement {
    Complex val;
    int row, col;
    < other fields >
    struct MatrixElement *
        nextRow, *nextCol;
};
```

• Simulation Method
  – We perform cycle-based simulation using a program-driven based on MINT that interprets program binaries and models configurable logic.
Case Studies

- Case 1: Architectural Adaptation for Latency Tolerance.
  - As the gap between processor and memory speed widens, prefetching is becoming increasingly important to tolerate the memory access latency.
  - Oblivious prefetching can degrade a program’s performance by saturating the bandwidth.

- The prefetcher implementation using programmable logic integrated with L1 cache.

Examples 1: Records spanning multiple cache lines and prefetches all fields of a matrix element structure whenever some field of the element is accessed.

Assume a cache line size is 32 bytes, a matrix element padded to 64 bytes and a single matrix storage block aligns at 64 bytes boundary.

Pseudocode of the prefetching scheme:

```c
/* Prefetch only if vAddr refers to the matrix */
GroupPrefetch(vAddr, pAddr, startBlock, endBlock) {
  if (startBlock <= vAddr && vAddr < endBlock) {
    /* Determine the prefetch address */
    if (pAddr & 0x20) ptrLoc = pAddr - 0x20;
    else ptrLoc = pAddr + 0x20;
    <Initiate transfer of line at ptrLoc to L1 cache>;
  }
}
```
Case Studies

- Example 2: Targets pointer fields that are likely to be traversed when their parent structures are accessed.

```c
/* Prefetch only if vAddr refers to the matrix */
PointerPrefetch(data, vAddr, startBlock, endBlock) {
    if (startBlock <= vAddr & vAddr < endBlock) {
        /* Get row pointer from returned cache line */
        ptrLoc = data [24]; /* row ptr offset = 24 */
        <Initiate transfer of elt at ptrloc to L1 cache>
    }
}
```

Case Studies

- Case 2: Architectural Adaptation for Bandwidth Reduction.
  - Use a sparse matrix-matrix multiply routine as the example.
  - Prefetch the whole rows and columns using pointer chasing in the memory module.
  - Packing/Gathering only the used fields of the matrix element structure.

Case Studies

- Implementation:

![Diagram ofProcessor and memory hierarchy]

```c
/* Row gather: pAddr is the start of a row */
Gather(pAddr) {
    chaseAddr = pAddr;
    while(chaseAddr) {
        forward chaseAddr->val
        forward chaseAddr->row
        chaseAddr=virtual-to-physical(chaseAddr->nextRow)
    }
}
```
Case Studies

• Pseudocode:

```c
translate(addr, pAddr, newAddr) {
    /* check if accessing start of a row */
    if(startRowAddr<=Addr && Addr<endRowAddr) {
        "Initiate prefetch and return row location";
        "Similarly for column roots ";
    ... /* Accessing packed rows */
    if(startPackedRows<=pAddr && pAddr<endPackedRows) {
        off = pAddr & 63; /* get field offset */
        if(off == 24) /* row ptr. at offset 24 */
            return pAddr + 64; /* synthesize next addr. */
        else /* Two fields at off1 and 2 are packed */
            if(off < offset) new_off = off-(off1-new_off1);
            else new_off = off-(off2-new_off2);
        return (pAddr>>6) * PACKED_SIZE + new_off; }
    /* Similarly for packed columns */
}
```

Case Studies

• Performance:

![Graph showing performance comparison]

Conclusion

• The increasing dominance of interconnect delays on system performance makes it practical to integrate programmable logic into all key system components.

• Such architectural adaptation can provide flexible mechanism for application-specific locality optimization.

• System co-design using this approach presents a way to utilize application-specific hardware much more effectively than co-processing architectures.