Software Abstractions to Improve the Scheduling of Data Center Workloads

Prof. Margaret Martonosi
Princeton University
Across many scales, the goal is similar: Max Performance at a Given Wattage

Fundamentally, this is a prediction and scheduling problem.
Across data centers

Request

Front-end

Data Center mirror 1

Data Center mirror 2

Data Center mirror 3

Internet
Our Prior Work: Prediction and Scheduling for Multi-data-center services

• Predict:
  – electricity prices, types, and availability
  – workload arrival rates and characteristics
  – network latencies...

• Use Simulated Annealing to determine near-optimal request distributions
  – (recalculate when predictions are too wrong, but that’s infrequent)
  – Can cut cost nearly 2X, compared to request distribution based on locality alone.
Within a data center

Request

Servers

... Servers

Map Phase

Reduce Phase

Servers

... Servers
Within a data center: Planning for power/performance requires...

- Predicting workload arrival rates
- Predicting types of requests
  - Read vs. write frequency
  - CPU intensity
  - Servers required
  - Long tail of latency (critical path)
  - Comm required between servers

Hard problem, but...
Can characterize computation times.
Can characterize communication flows.
Within a single processor chip:
Increasing corecounts and heterogeneity
=> communication matters here, too.
Questions

• Communication is increasingly center to each of these scenarios, but software often exposes too little about communication patterns and requirements.
• Can we create graph-based expressions of program computation/communication needs
  – For dynamic mapping and scheduling
  – For locality optimizations
  – For critical path analysis
• What is the appropriate approach for within-server and within-data-center optimizations?
  – Granularity?
  – Semantics?

Thus far:
• Started to explore graph-based approaches on-chip.
Next steps:
• Deepen on-chip efforts.
• Expand to coarser-grained multi-server scenarios
Proposed Approach and Semantics

**System-level ISA Graph**
- Nodes are chunks of computation
  - [Much] coarser-grained than individual instructions
- Data dependences form edges
- Initial schedule mapped out statically
- Dynamic refinements based on profiling or ongoing characterizations and annotations

**Semantics**
- **Within nodes: no internal side effects**
  - only computation or local (scratchpad) refs
- All shared (inter-node) communication is explicitly marked as data dependences along edges
- **Optional**: Other information can also be annotated onto the graph.
  - Profile data
  - Application preferences and constraints
- With these rules, graph chunks can be mapped and managed at granularity of 100’s or 1000’s of instructions or larger
float data[], sqr[];

void main()
{
    int i;
    init(data);

    for (i = 0; i < MAX; i++)
        sqr[i] = data[i] * data[i];

    output(square);
    return;
}
Example: SISA-Informed Memory Management & Inter-Core Communication

• Producer-consumer relationships are common in parallel code, and yet not well-optimized by “traditional” memory hierarchies…

[Diagram of producer and consumer with data transfer]

- Producer
- Consumer
- Data[i]…

[Diagram of memory hierarchy]

Core 0
- L1$
- TLB
- Core 1
- L1$
- TLB
- L2$
- Main Memory
Example: SISA-Informed Memory Management & Inter-Core Communication

• Producer-consumer relationships are common in parallel code, and yet not well-optimized by “traditional” memory hierarchies...

• Inefficient exploitation of program knowledge:
  – Granularity of data
  – Communication relationship
  – Begin/end of data usage
Example: SISA-Informed Memory Management & Inter-Core Communication

• Producer-consumer relationships are common in parallel code, and yet not well-optimized by “traditional” memory hierarchies...

• Inefficient exploitation of program knowledge:
  – Granularity of data
  – Communication relationship
  – Begin/end of data usage
Example: SISA-Informed Memory Management & Inter-Core Communication

- SISA graphs provide explicit information about memory and communication
- Idea 1: Use SISA to discern producer-consumer relationships
  - Push data to consumer cores proactively
  - Manage comm schedule to reduce power dissipation on non-critical edges
  - Self-invalidate data at producer core when write occurs.
- Idea 2: Use SISA to manage last-level cache.
  - Use SISA graph to predict low-reuse items and insert them in LRU position
Producer-Consumer Stressmark: Results

• Thrashing stress test results measured using Simics/FeS2 simulator
• Graph-guided “move to LRU” technique:
  • Improves performance
    – Cut L1 cache miss rate nearly in half
• Improves power too...
Example 2: Heterogeneous Parallelism: IPSec Packet Forwarding Pipeline

- Heterogeneity improves both overall performance and perf-per-Watt
- Thus far: Basic mapping to Intel QuickIA board
- Next steps: Adaptive mapping based on workload and accelerator availability.
Next Steps

• Automatic OpenCL to SISA translator
• Coarsen SISA graphs to granularity of per-server functions in distributed data center apps.
• Use hardware performance counters and other available measurement approaches to guide dynamic remappings
  – On-Chip: Between accelerators or cores
  – Between Servers:
    • Migrate workloads to improve performance or power characteristics
    • Better data staging.
Takeaway message

• Today’s Cloud Computing needs agile power/performance tradeoffs.
  – Multi-granular; Parallelism; Heterogeneity

• Current programming and execution models give insufficient view of communication/computation relationships
  – Across multiple servers
  – Across multiple processors within a single server

• Our Work: Graph-based abstractions with profiling characterizations to guide dynamic heuristic scheduling or periodic simulated annealing-based scheduling runs.
Software Abstractions to Improve the Scheduling of Data Center Workloads

Prof. Margaret Martonosi
Princeton University

+ Students:
Ozlem Bilgir, Wenhao Jia, Dan Lustig,
Carole-Jean Wu, Yavuz Yetim