Scribe notes for Oct 16

Overview:

Finished Discussion on SYMPO
- Code Generator Description
- Genetic Algorithm Description
- Results compared to handmade power viruses

Started Discusion on Data Center Paper
- Key points of the paper
- Items Left Out

Other Notes:
- SPEC CPU 2006 benchmarks available for your project just need to ask the TA about it

Continuation of SYMPO Paper

There are many x86 specific power viruses but each is generally tailored for a specific architecture (except Mprime) In this instance there was a unique power virus for each AMD Processor

In the SPEC CPU 2006 Benchmarks Games and Povray have the highest power usage of the Spec CPU benchmark but they are still less than the hand made power viruses (60 Watts for spec compared to 70+ Watts for power virus)

- This can be important for helping find the highest power mark that a real program can reach

SYMPO creates power virus given various non-architecture specific parameters for the program ie instruction mix, memory stride, branch predictability, register dependency distance

Sends it to a simulator or model to determine the power used

Genetic Algorithm:

- A genetic algorithm (though other methods are possible) creates new configurations for the code generator and uses the feedback from the previous results to improve each successive attempt

- SYMPO uses a genetic algorithm to create new configuration files where each parameter in the configuration file can be thought of as a knob that the GA will adjust in-order to create new individuals

- IBM SNAP was used as the GA but its proprietary GAUL is a good open source one

  Key Terms as defined for SYMPO:
  - Individuals are the synthetic workload
  - Genes are the parameters for the workload
  - Crossover creates a new individual by selecting parameters from two individuals
  - Fitness Function is the power from the simulator or model
  - Mutation rate chance that a gene is changed to a completely random value

- Different GA implementations have different heuristics that are often tunable that make two implementations very different in terms of number of generations it takes to get a specified result

Code Generation:

(Not all steps required to create a power virus but this approach can be used for other purposes as well like creating similar performing synthetic programs based off of characteristics of a real program)

- The programs start with a template that usually involves a loop or an inner and outer loop with basic blocks inside (the template is ready ahead of time and the rest just inserted in

- Fix the number of basic blocks

- Choose an instruction type for each opening in each basic block fo based on the instruction mix weights provided by the configuration file

- Create the conditional jumps between the conditional blocks (ie loops or forward branches) and bias them according to the configuration parameters (allows the creation of various branches wih different predicatability

- Assign sources and destination operands based off of your configuration that specifies dependency distances for registers

- Assign memory accesses based off of the stride patterns described by the configuration file

SYMPO Results:

- Other computer learning methods were tested but Genetic Algorithms performed the best and were fairly easy to use

- SYMPO was able to beat the tailored power viruses and more importantly does so without requiring user interaction (Times varied for how long it took to create a better virus but ussually between 1-3 days.

- A GA and many of the similar other machine learning methods can run into problems due to important properties not being adjustable by the knobs or a combination of them.   A real program may have properties that are not accurately modeled/adjustable by the code generator.

- The starting point for the genetic algorithm can be anywhere in the search space but will take variable amounts of time to converge on its best results. In this case it can take days to create viruses that beat Mprime but most of that is due to the simulation time.

- Eventually the results will stop improving or at least less quickly or may get stuck in a local minimum for a while before improving again. Can be hard to know when to stop without having a defined target but may not reach it either

- Different high performing power viruses generated by the code generator looked very different.

Understanding and Designing New Server Architectures
for Emerging Warehouse-Computing Environments from 2008:

Tried new discussion format for this paper so notes will be a bit less organized

This was not typical ISCA paper: More ambitious in terms of scope and easier to read but less on the details

Key Points:

- Using commodity parts vs actual server system can give a better tradoff for performance vs total cost of ownership
    - Volume reduces costs
    - Don't necessarily need all of the capability of a larger system
    - Many designed with power in mind

- They created their own benchmarksand tried out 6 different types of configurations with performance/total cost of ownership being the primary metric

- Proposed new methods of cooling and structuring memory/architectural changes to improve performance

- System level exploration of the issue which had not been seen at that time

Problems discussed in the paper:

- TOC is half buying the system and half cooling and power and infrastructure (operating costs)

- Issues with power inefficiency
    - Data center carbon footprint is around 18% for the world

- Over provisioning is a significant issue (buying for worst-case)

- Laptop Hardrives may use less power

- Using Flash Memory for LLC misses

A few of the Insights hinted at in the paper but not fully described:

- Data centers are often setup near natural resources and often kept in cooler regions to reduce

- Most buildings have a limit on power that can be brought in and its expensive to retrofit. Creates a hard limit on how many computers can be running at once. ECE was a good example, start-over from scratch then retrofit used in the justification

The Benchmarks:

Wanted to find what systems provide the most performance for power used

-Ytube and mapreduce are IO bound and do well on some of the lower end systems

- mail and search did worse on the lower end systems and are compute bound and have low off-chip bandwidth requirements

New Cooling and architecture ideas introduced in the paper:

- Convert cooling from a Rack based cooling solution to a blade based solution

(Side not on rack layout)
Racks are divided into units with varying for factors
Blades are a more like a box with multiple slots that you can insert additional blade processors/boards (computation, memory, etc) upto the limit of the numeber of available slots for that box

- Had local memory for each blade and a separate blade that has memory to be shared by each processing blade

- Using Laptop harddrives for storage (for low power usage)

- Using Flash after last level memory:
- Recommended NAND FLASH instead of NOR FLASH (NAND is cheaper and denser but must be written in blocks)
-NAND writes faster NOR reads faster

Below are some of the items not discussed in the paper but are of significant consequence:

- Assumed that the various alternative setups could scale out and would not incur too much overhead but did not actually prove the point.
-Bandwidth and inter-core communication may not scale well as you increase the number of servers

- Software support is included for the servers and the programs that are designed to run on them but may not have the same support for the other solutions

-Uptime and reliability not included in the cost model or discussed

-Security also not included in considerations or cost models

- FLASH lifetime cycles not really covered considering that each one won't degrade at the same rate

Where it stands today:

- Some companies are developing servers (micro servers) based off of embedded systems

- Virtualized OS and Custom racks designed to address over-provisioning issues

- Cloudsuite benchmark for these systems

- EPFL scale out processor simulator

Scale out (more processors) vs Scale up (bigger processors)

- Execution driven simulation doesn't work well as you dig deeper in the hierarchy towards systems (like IO) that are not accessed frequently compared to the rest of the events the simulator has to run (who would want to simulate a billion instructions just to simulate a handful of IO events of interest.