Adaptive Power Efficiency: Runtime System Approach with Hardware Support

Ehsan Totoni

Advisor:
Laxmikant V. Kale
{totoni2, kale}@illinois.edu

Adaptive Power Efficiency

Power efficiency is the main challenge for HPC
- 20MW goal for exascale

Systems are designed for the general case
- Processor architecture, network topology etc.

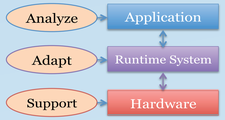
But applications are diverse
- Very different properties

Not all system resources are used efficiently

Cache inefficiency example:
- No locality in many "Big Data" applications
- But caches are consuming significant power anyways!

Network inefficiency example:
- Example: No communication in data-parallel applications
- But network links are always on and consume power!

Solution: adaptive power efficiency
- Adaptive runtime system with hardware support
- Based on application analysis



Related Work

Hardware based reconfiguration schemes
- Hardware complexity, energy overhead
- Hard to predict application behavior (small window)

Compiler based schemes
- Many assumptions for analysis (Simple affine nested loops), (Simple array indices)
- Not feasible for real applications

References:
Using an Adaptive HPC Runtime System to Reconfigure the Cache Hierarchy, Totoni et al., SC'14, 2014.
Power Management of Extreme-scale Networks with On/off Links in Runtime Systems, Totoni et al., TOPC, 2013.

http://charm.cs.illinois.edu

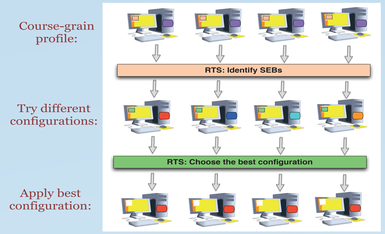
Runtime System Approach

Runtime system (RTS) monitors application
- Sequential computations between comm. calls (SEBs)
- HPC applications are iterative, same pattern repeats

RTS tries different configurations
- E.g. number of cache ways on/off for each SEB
- In parallel, since different typically processors do the same thing in SPMD paradigm

RTS applies best configuration
- E.g.: Turn on/off ways
- Monitor, re-evaluate regularly

Very low overhead: done only once
- HPC applications run for hours!

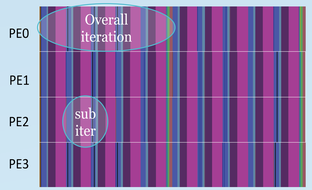


HPC Applications are iterative and predictable
- Hierarchical iterative pattern in MILC:

Can be represented by a regular expression
- MILC's pattern:

$$((a_0a_1a_2a_3)^5b_0b_1b_2b_3)^*$$

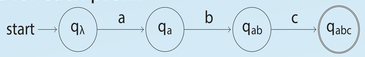
- Each symbol is sequential computation between communication calls (SEB)



RTS builds DFA in profiling phase: formal language theory

Learning a regular language from text

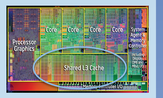
Solution: Prefix Tree Acceptor (PTA)
- One state for each prefix



DFA for *(abc)**

Cache Adaptation

Caches consume a large fraction of processor's power
- 40% in POWER7, even with many power optimizations
- Fixed design, but applications are different



Cache Inefficiency: NAMD on Blue Waters
- HIV simulations, 64 million atoms
- 48 bytes atom state (position & velocity) + some transient data
- Assuming 400 bytes/atom, 25.6 GB
- 4000 Cray-XE nodes
- 32 MB of L2 and 32 MB L3 each -> 256 GB of cache!
- 90% of cache capacity not unused (there is nothing wrong with NAMD!)
- 16 days wall clock time, not much use of caches... Huge waste!

Solution: Turning off cache ways adaptively
- Two main issues: (finding best configuration), (predicting application's future)
- Our RTS approach handles both issues effectively

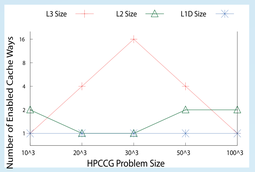
Evaluation results:
- Mantevo mini-apps on SESC simulator
- Best configuration depends on: application type, input size
- Not easy to predict without testing

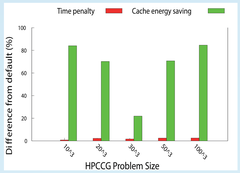
67% cache energy saving on average
- 28% of whole processor, 2.4% performance penalty

Format: <ways turned on>/<total number of ways>

Mini-App	L1D	L1I	L2	L3
miniMD	2/4	1/2	2/8	1/16
CloverLeaf	1/4	1/2	2/8	16/16
HPCCG	1/4	1/2	2/8	16/16

Adaption to problem size is necessary:
- Larger sizes cannot exploit caches efficiently

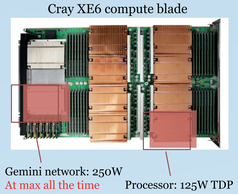




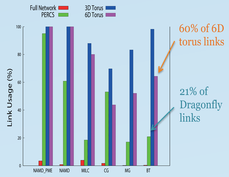
Network Adaptation

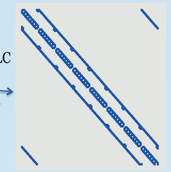
Networks consume ~30% of system's power
- Not energy proportional- near peak all the time

Blue Waters example
- ~35kW running rack, ~15kW when idle due to network
- Not all applications use all the network's capabilities



Networks are designed for worst cases (e.g. FFT)
- But application communication patterns are different
- Nearest neighbor most common
- A large fraction of link unused





Solution: RTS turns links off adaptively
- 81.5% of links can be turned off for common applications
- On/off scheduling saves even more

Process Variation Heterogeneity

Transistors on chip are different
- Cores of a manycore have different speeds and power
- Worse in future generations, especially low voltage

RTS adaptation is required
- For each application, how many and which cores to use
- Scheduling framework using performance and power models
- Based on integer linear programming (ILP)

Ask me for more info!

Conclusion

Runtimes systems can improve power efficiency significantly
- Monitor application and hardware and adapt
- Very low overhead, minor hardware support