REDUCING POWER CONSUMPTION OF PARALLEL APPLICATIONS

Daniele De Sensi
OUTLINE

▷ Hardware support
▷ Solutions
▷ Hands-On
▷ Conclusions
Hardware support
POWER AND ENERGY

**Power**

- Instantaneous consumption
- Expressed in Watts (W)

**Energy**

- Power x Time
- Expressed in Joules (J)
POWER CONSUMPTION

\[ P_{CPU} = P_{Static} + P_{Dynamic} \]

- Accounted also when the system is idle
- Accounted only when the system is computing
POWER CONSUMPTION

\[ P_{CPU} = V \cdot I_{leak} \]

Reduced by reducing voltage
POWER CONSUMPTION

\[ P_{CPU} = V \cdot I_{\text{leak}} + \alpha \]

Reduced by changing algorithm
POWER CONSUMPTION

\[ P_{CPU} = V \cdot I_{leak} + \alpha \cdot N \]

Reduced by reducing the number of cores used
POWER CONSUMPTION

\[ P_{CPU} = V \cdot I_{\text{leak}} + \alpha \cdot N \cdot f \cdot V^2 \]

Reduced more than linearly by linearly reducing the frequency
C-STATES

▷ C0 - Active mode
   Core is active and executing instructions (P-states)

▷ C1
   Core is not executing instructions, scales down frequency

▷ C1E
   Scales down voltage

▷ C3
   Core flushes L1 and L2 caches to L3 cache. Clocks are stopped

▷ C6
   Architectural state saved to a dedicated SRAM. Voltage reduced to 0V

▷ C7
   L3 cache flushed

- Static power
+ Wake up latency
P-STATES (DVFS)

▷ P0 - Highest frequency
▷ P1
... 
▷ Pn - Lowest frequency

- Dynamic power
+ Execution time
C states are present both on CPU, cores and contexts.

A unit can enter a specific C-state only if all its subunits are in that state or in a deeper state (e.g. CPU with 2 cores. Core 0 is in C2 and Core 1 is in C6 -> CPU can enter C2).

One clock for an entire CPU -> All the cores on the CPU must be in the same P-state.
HETEROGENEOUS HARDWARE

- CPU, GPU, FPGA, DSP, coprocessors, etc...
- Differ for kind of computation they can execute and for power consumption
- Different types of cores on same machine (e.g. ARM’s big. LITTLE)
Using an Adaptive HPC Runtime System to Reconfigure the Cache Hierarchy

Ehsan Totoni, Josep Torrellas, Laxmikant V. Kale

Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
E-mail: {totoni2, torrella, kale}@illinois.edu

Power Management of Extreme-scale Networks with On/Off Links in Runtime Systems

Ehsan Totoni, University of Illinois at Urbana-Champaign
Nikhil Jain, University of Illinois at Urbana-Champaign
Laxmikant V. Kale, University of Illinois at Urbana-Champaign
RESUMING

\[ P_{CPU} = V \cdot I_{leak} + \alpha \cdot N \cdot f \cdot V^2 \]

- **C-States** (Idle)
- **Implementation**
- **P-States** (Frequency)

Different architecture
Solutions
RESUMING

\[ P_{CPU} = V \cdot I_{\text{leak}} + \alpha \cdot N \cdot f \cdot V^2 \]

\[ T \propto \frac{1}{N} \]

\[ T \propto \frac{1}{f} \]
POWER OR PERFORMANCES?

Performance

Power
USE CASES

Follow trends to decrease resources when there is no work to do

Power capping to avoid high temperatures

Execution time capping for non time-critical computations
Follow trends
PARALLEL APPLICATIONS
PARALLEL APPLICATIONS

Power reduced by:
- Changing the number of threads (cores)

And/Or
- Changing the frequency of the cores

Wider range of possibilities and finer grained control w.r.t. frequencies only
PARALLEL APPLICATIONS

Energy driven adaptivity in stream parallel computations

Marco Danelutto, Daniele De Sensi and Massimo Torquati
Dept. of Computer Science, Univ. of Pisa
Email: daniele.desensi@for.unipi.it {marcod,torquati}@di.unipi.it

- Decrease number of threads and frequency when the system is underutilized
- Increase number of threads and frequency when the system is overutilized
- What to decrease and how much to decrease?
ALGORITHM

Utilization

Frequency (GHz)

Number of workers

1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0

1 2 3 4 5 6 7 8 9 10 11 12 13 14

Utilization
RESULTS

Bandwidth and reconfigurations comparison

Bandwidth (Pkts/Sec)  Workers x Frequency

Time (minutes)
RUNTIME SUPPORT VS. OS GOVERNOR

![Graph showing runtime support vs. OS governor](image-url)
Power capping
Power Capping Via Forced Idleness

Anshul Gandhi
Carnegie Mellon University
anshulg@cs.cmu.edu

Mor Harchol-Balter
Carnegie Mellon University
harchol@cs.cmu.edu

Rajarshi Das
IBM Research
rajarshi@us.ibm.com

Jeffrey O. Kephart
IBM Research
kephart@us.ibm.com

Charles Lefurgy
IBM Research
leurgy@us.ibm.com

▷ Forced “race-to-idle”. Processor goes idle also if has work to do
▷ Budget = 170 Watts. Alternates between 120 Watts (Idle) and 220 Watts (Max frequency) spending half of the time in each state.
▷ Guarantees on maximum execution time can be provided (Formulation that relates budget to performance)
1. Pack more threads on less cores
2. Machine learning to predict behaviour
THREAD MAPPING
Time capping
RACE-TO-IDLE VS. EXPLOITING SLACK

Race to idle
RACE-TO-IDLE VS. EXPLOITING SLACK

Race to idle

Exploiting Slack
Our results show that on our Pentium-based system, it is energy efficient to run at the highest performance state, and on the PowerPC-based system, at a lower frequency when voltage is kept constant.

Critical power slope can be used as a metric to understand how efficient the system is in regards to idle state relative to active state.
Hands on
Cpu speed from cpufreq 2700.00Mhz

cpufreq might be wrong if cpufreq is enabled. To guess correctly try estimating via tsc
Linux’s inbuilt cpu_khz code emulated now
True Frequency (without accounting Turbo) 2700 MHz
  CPU Multiplier 27x || Bus clock frequency (BCLK) 100.00 MHz

Socket [0] - [physical cores=8, logical cores=16, max online cores ever=8]
  TURBO ENABLED on 8 Cores, Hyper Threading ON
  True Frequency 2800.00 MHz (100.00 x [28])
  Max TURBO Multiplier (if Enabled) with 1/2/3/4/5/6 Cores is 35x/35x/34x/32x/32x/32x
  Current Frequency 3499.87 MHz [100.00 x 35.00] (Max of below)

<table>
<thead>
<tr>
<th>Core [core-id]</th>
<th>Actual Freq (Mult.)</th>
<th>C0%</th>
<th>Halt(C1)%</th>
<th>C3 %</th>
<th>C6 %</th>
<th>C7 %</th>
<th>Temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core 1 [0]:</td>
<td>3499.87 (35.00x)</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>48</td>
</tr>
<tr>
<td>Core 2 [1]:</td>
<td>3445.10 (34.45x)</td>
<td>1</td>
<td>0.447</td>
<td>0</td>
<td>99.3</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Core 3 [2]:</td>
<td>3436.41 (34.96x)</td>
<td>1</td>
<td>0.501</td>
<td>0</td>
<td>98.7</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>Core 4 [3]:</td>
<td>3421.85 (34.22x)</td>
<td>0</td>
<td>0.0182</td>
<td>0</td>
<td>100</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>Core 5 [4]:</td>
<td>3439.32 (34.30x)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>38</td>
</tr>
<tr>
<td>Core 6 [5]:</td>
<td>3425.83 (34.26x)</td>
<td>0</td>
<td>0.00972</td>
<td>0</td>
<td>100</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>Core 7 [6]:</td>
<td>3429.16 (34.29x)</td>
<td>0</td>
<td>0.0153</td>
<td>0</td>
<td>100</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>Core 8 [7]:</td>
<td>3395.37 (33.95x)</td>
<td>0</td>
<td>0.00945</td>
<td>0</td>
<td>100</td>
<td>37</td>
<td></td>
</tr>
</tbody>
</table>

C1 = Processor running with halts (States >C0 are power saver)
C3 = Cores running with PLL turned off and core cache turned off
C6 = Everything in C3 + core state saved to last level cache
  Above values in table are in percentage over the last 1 sec
  [core-id] refers to core-id number in /proc/cpuinfo
  'Garbage Values' message printed when garbage values are read
  Ctrl+C to exit
C-STATES

○ Idle loop executed when the Linux scheduler has no thread to run.
○ A 'governor' decides whether to continue in current C-state or to move to a different state.
  ▷ Ladder: One step at a time towards the deepest C-state
  ▷ Menu: Can skip intermediate C-states if it thinks it’s worthwhile
C-STATES

Current governor in: /sys/devices/system/cpu/cpu0/cpuidle/current_governor_ro

Management of Idle States: /sys/devices/system/cpu/cpu*/cpuidle/state*/

> ls /sys/devices/system/cpu/cpu0/cpuidle/
  state0 state1 state2 state3 state4
> ls /sys/devices/system/cpu/cpu0/cpuidle/state1/
  desc disable latency name power time usage
> sudo echo 1 > /sys/devices/system/cpu/cpu0/cpuidle/state1/disable
P-STATES (DVFS)

May be managed by means of Governors:

- Performance (P0)
- Powersave (Pn)
- Ondemand, Conservative
  (parameters can be tuned by modifying files in /sys/devices/system/cpu/cpufreq/ondemand/)
- Userspace (controlled by the user)
P-STATES (DVFS)

Controlled by accessing files in: /sys/devices/system/cpu/cpu*/cpufreq

> cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_governors
  conservative userspace powersave ondemand performance
> echo userspace > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
> cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_freq
  2400000 2300000 2200000 2100000 2000000 1900000 1800000 1700000
  1600000 1500000 1400000 1300000 1200000
> echo 1500000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
> ls /sys/devices/system/cpu/cpu0/cpufreq/
  affected_cpus                   scaling_cur_freq
  cpuinfo_transition_latency      scaling_governor
  scaling_available_frequencies   scaling_max_freq
  scaling_available_governors     scaling_min_freq
> cpufreq-info
analyzing CPU 30:
  driver: acpi-cpufreq
  CPUs which run at the same hardware frequency: 8 9 10 11 12 13 14 15
  24 25 26 27 28 29 30 31
  CPUs which need to have their frequency coordinated by software: 30
  maximum transition latency: 10.0 us.
  hardware limits: 1.20 GHz - 2.00 GHz
  available frequency steps: 2.00 GHz, 1.90 GHz, 1.80 GHz, 1.70 GHz,
  1.60 GHz, 1.50 GHz, 1.40 GHz, 1.30 GHz, 1.20 GHz
  available cpufreq governors: ondemand, userspace, performance
  current policy: frequency should be within 1.20 GHz and 2.00 GHz.
    The governor "userspace" may decide which speed to
    use within this range.
  current CPU frequency is 2.00 GHz.
> cpufreq-set -c 30 -g userspace
> cpufreq-set -c 30 -f 1.90GHz
Mammut m;

vector<Domain*> d = m.getInstanceCpuFreq()->getDomains();

d[0]->setGovernor(GOVERNOR_USERSPACE);

d[0]->setFrequency(3000000);

d[1]->setGovernor(GOVERNOR_ONDEMAND);

1. https://github.com/DanieleDeSensi/Mammut
Mammut m(CommunicatorTcp(address, port));
VirtualCore* vc = m.getInstanceTopology()->getVirtualCore(0);
vector<VirtualCoreIdleLevel*> levels = vc->getIdleLevels();
if(levels[2]->getExitLatency() > MAX_LATENCY){
    levels[2]->disable();
}
Mammut m;
Energy* e = m.getInstanceEnergy();
CounterCpu* counter = e->getCountersCpu()[0];
JoulesCpu before = c->getJoules();

// Execute code
JoulesCpu after = c->getJoules();
cout << "Joules consumed: " << (after - before) << endl;

1. https://github.com/DanieleDeSensi/Mammut
Conclusions
ADAPTIVE COMPUTATIONS

▷ Optimal solution depends on different factors:
  ○ Hardware
  ○ Application
  ○ Input
▷ Wrong decisions ⇒ Catastrophic effects on performance and power
▷ Optimal solution can change through time
▷ Application programmer can’t afford this burden
▷ Runtime support that deals with these problems
STRUCTURED PROGRAMMING

Parametric wrt number of threads

Well defined cost model

Hides complexity

By changing the number of threads we don't change semantics

We can predict the behaviour for different parameters

We can hide mechanisms management
RUNTIME SUPPORT

```c
struct Emitter: ff_node{
    void* svc(void*){
        // Generate a data element t
        // (e.g. by reading a packet from the network)
        return t;
    }
};

struct Worker: ff_node{
    void* svc(void* t)
        // Compute something on t
        return t;
    }
};

int main (int argc, char* argv[]){
    vector<Worker*> w;
    for(uint i = 0; i < MAX_NUM_CORES; i++){
        w.push_back(new Worker);
    }
    ff_farm<> f(workers);
    f.add_emitter(new Emitter);

    AdaptivityParameters ap("parameters.xml");
    ManagerFarm amf(&f, ap);
    amf.start();
    amf.join();
}
```
Thanks

desensi@di.unipi.it