When Is Energy ≠ Time?: Understanding Energy Scaling with eAudit

Sudhakar Yalamanchili and Eric Anger
School of Electrical and Computer Engineering
Georgia Institute of Technology

Scaling Performance and Power

\[ \text{Perf} \left( \frac{\text{ops}}{s} \right) = \text{Power} (W) \times \text{Efficiency} \left( \frac{\text{ops}}{\text{joule}} \right) \]

W. J. Dally, Keynote IITC 2012

- Increasing performance requires increasing system scale → parallelism
- Scaling parallelism does not effect energy and time in the same way!

<table>
<thead>
<tr>
<th></th>
<th>Current</th>
<th>Exascale</th>
<th>Growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>3.1M</td>
<td>1B</td>
<td>~300x</td>
</tr>
<tr>
<td>Power</td>
<td>17.8MW</td>
<td>20-40MW</td>
<td>~1.5-2.5x</td>
</tr>
</tbody>
</table>
Energy Scaling vs. Time Scaling

Developers should understand how both energy and time scale with parallelism.

Relationship between energy scaling and time scaling?
- When are they not the same? Tradeoffs?
- Drive the development of diagnostic tools

Why Is This an Application/Algorithm Problem?
Managing Thermal Capacity: Thermal Coupling

- Significant rise in temperature of the idle component due to thermal coupling and pollution
- CPU cores consume thermal headroom more rapidly (4X faster)
- GPUs sustain power boosts longer!
- Better management for 10%-40% gains in measured energy efficiency are possible
- Power management ≠ thermal management


Managing Power Capacity

- Dynamic Demand
- Power Sensitivity → Energy Efficiency
- Design Space² (BFS)

2. A. McLaughlin et al., “A Power Characterization and Management of GPU Graph Traversal,” ADMS 2014
Shift in the Balance Point

Balance plane for performance and energy

- Relative energy costs of compute and memory access
- Relative ops/byte demand of application

Up to 36% power savings with a maximum performance loss of 3.6%


Understanding Application-Level Energy Scaling

Different implementations of BFS on different input data sets

- Note the variance of energy dissipation across different implementations of the same function
- Challenge: How do we understand the energy implications of our decisions? Algorithms, data structures, etc.
When Is Energy ≠ Time?

You can hide latency but not energy!

Energy Components

- **Static Energy**
  - Scales with execution time
  - E.g., leakage energy
    - Technology dependent
  - Idle energy consumption (OS, I/O, etc.)
  
  \[
  \text{Energy overhead} = \begin{cases} 
  \text{Constant under ideal strong scaling} \\
  \text{Grows under weak scaling}
  \end{cases}
  \]

- **Dynamic Energy**
  - Scales with work
  - Independent of time (strong scaling)
  - Energy required to solve problem

Energy scaling is limited by the fraction corresponding to static energy
Amdahl’s View of Energy Scaling

Base case is a single core executing a serial algorithm

- Ideally energy speedup tracks time speedup (no idle energy)

\[
S_e = \frac{1}{1 - s + \frac{s}{S_s}} \rightarrow \text{static fraction}
\]

\[
S_t = \frac{1}{1 - f + \frac{f}{p}} \rightarrow \text{serial fraction}
\]

\[
\eta_t = \frac{S_t}{p} \quad s = \frac{E_s}{E_s + E_d} \rightarrow \text{dynamic energy}
\]

Impact of Concurrency

- In time, strong scaling is limited by the serial fraction
- When it is small, large benefit from strong scaling
- In energy, strong scaling is limited by the static fraction
- Static fraction is multiplicative penalty in addition of the serial fraction

\[
S_e = \frac{1}{(1 - f \times s) + p \times f \times s}
\]
Energy Scaling vs. Time Scaling

Impact of time on energy scaling is a function of the static fraction

\[ \text{Time \_speedup} \geq \text{Energy \_speedup} \geq 1.0 \]

Runtime vs. Design Time?
Energy Auditing: eAudit

- Application energy auditing tool
  - *Function-level* attribution
- Diagnose application energy consumption behavior
- Provide actionable information to steer energy optimization

**Example output**

<table>
<thead>
<tr>
<th>Name</th>
<th>Energy</th>
<th>Time</th>
<th>Instructions</th>
<th>% Energy</th>
<th>% Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>generate_matrix(hidden int, int, HPC Sparse Matrix_STRUCT*, double*, double**, double*) at ??</td>
<td>4.3672</td>
<td>0.115</td>
<td>372291000</td>
<td>54.9705</td>
<td>12.3658</td>
</tr>
<tr>
<td>HPC sparse/HPC Sparse Matrix_STRUCT*, double const*, double*) at HPC sparsemv.cpp ?? at ??</td>
<td>2.3132</td>
<td>0.511</td>
<td>1214509</td>
<td>29.12098</td>
<td>54.9462</td>
</tr>
<tr>
<td>wxspxy(int, double, double const*, double, double const*, double*) at wxspxy.cpp</td>
<td>1.0075</td>
<td>0.238</td>
<td>4633900000</td>
<td>12.06337</td>
<td>25.5914</td>
</tr>
<tr>
<td>ddot(int, double const*, double const*, double*, double**, double*) at ddot.cpp</td>
<td>0.1348</td>
<td>0.037</td>
<td>67981800</td>
<td>1.69754</td>
<td>3.97849</td>
</tr>
</tbody>
</table>

**eAudit Implementation**

- Sampling-based measurements, similar to *gprof*
- RAPL limited to all cores on package: future versions should expose per-core counters
Model does not include variations to energy due to growth in work
- Sharing, locking, barriers, etc.
- Typically a function of # threads

Prefetching decreases time and energy, but not my same degree
- Reduction in time $\rightarrow$ reduction in static energy
- Speculative memory traffic $\rightarrow$ increase in dynamic energy
Need Better Energy Instrumentation!

Scaling Across Sockets

- eAudit demonstrated at board-level
- Next steps:
  - Add network energy models
    - system-level application energy audit

<table>
<thead>
<tr>
<th>Name</th>
<th>Package Energy Speedup</th>
<th>Package + Memory Energy Speedup</th>
<th>Time Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPC_sparsemv</td>
<td>1.03</td>
<td>1.10</td>
<td>1.94</td>
</tr>
<tr>
<td>waxpby</td>
<td>1.32</td>
<td>1.39</td>
<td>2.14</td>
</tr>
<tr>
<td>ddot</td>
<td>1.46</td>
<td>1.48</td>
<td>2.30</td>
</tr>
<tr>
<td>generate_matrix</td>
<td>0.59</td>
<td>0.62</td>
<td>1.00</td>
</tr>
<tr>
<td>sys</td>
<td>0.14</td>
<td>0.15</td>
<td>0.24</td>
</tr>
</tbody>
</table>
Extending eAudit: The lwperf Library

```c
function foo(){
    lwperf_log("region1");
    // Computation of interest
    lwperf_stop("region1");

    // Other work...

    lwperf_log("region2");
    // Another computation
    lwperf_stop("region2");
    // ...
}
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Energy (j)</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>region1</td>
<td>133.2</td>
<td>1.1</td>
</tr>
<tr>
<td>region2</td>
<td>552.7</td>
<td>3.3</td>
</tr>
</tbody>
</table>

- Enables measurements across regions of interest
- Integration with Eiger
  - For further analysis/modeling
  - One measurement per log/stop pair

Summary

- Application design should take energy behavior into consideration to reach performance goals
- Characterize energy scaling as a function of static and dynamic energy
  - Time scaling only improves static energy
- Basis for eAudit, an energy measurement and analysis tool

\[ S_e = \frac{1}{(1 - s) + \frac{s}{\eta_s}} \]

\begin{align*}
S_e & = \frac{1}{(1 - s) + \frac{s}{\eta_s}} \\
\eta_s & = \text{static fraction}
\end{align*}

eAudit available: github.com/gtcasl/eaudit
Energy Scaling Model

Energy with $p$ cores

$$E_p = (E_s \times p) \times \frac{T_p}{T_1} + E_d$$

$$E_p = \frac{E_s}{\eta_t} + E_d$$

Scaling static power with time

Time and Energy Efficiency

$$\eta_t = \frac{T_1}{T_p \times p}$$

$$\eta_e = \frac{E_d}{E_s + E_d}$$

Energy Speedup

$$S_e = \frac{1}{\frac{1}{\eta_t} + \eta_e}$$

$$s = 1 - \eta_e$$

Base case is a single core executing a serial algorithm