Investigating the Energy Efficiency of Iterative Sparse Linear System Solvers

Enrique S. Quintana-Ortí
The CG Method

- Why?
  - CG is key for the solution of s.p.d. sparse linear systems
  - CG boils down to sparse matrix-vector product (SpMV), a crucial kernel for many other scientific apps.
  - SpMV presents a memory-bound, irregular data access that reflects real-world apps.

HPCG benchmark (J. Dongarra & M. Heroux)!
The CG Method


def CG_method(r_0, p_0, x_0, \sigma_0, \tau_0; j := 0)
    
    while (\tau_j > \tau_{\text{max}})
        v_j := A p_j
        \alpha_j := \sigma_j / p_j^T v_j
        x_{j+1} := x_j + \alpha_j p_j
        r_{j+1} := I
        \zeta_j := r_{j+1}^T r_{j+1}
        \beta_j := \zeta_j / \sigma_j
        \sigma_{j+1} := \zeta_j
        p_{j+1} := z_j + \beta_j p_j
        \tau_{j+1} := \| r_{j+1} \|_2 = \sqrt{\zeta_j}
        j := j + 1
    
    Loop for iterative CG solver
    O1. SPMV
    O2. DOT
    O3. AXPY
    O5. DOT product
    O6. Scalar op
    O7. Scalar op
    O8. XPAY (AXPY-like)
    O9. Vector 2-norm (in practice, sqrt)

Memory-bounded kernels!
Outline

- Characterizing architectures via CG
- Energy efficiency of PCG
- Energy saving for multi-core and GPU servers
Characterizing architectures via CG Setup

- Performance of CG depends on
  - Target architecture: frequency-voltage setting, #cores, arithmetic floating-point precision, etc.
  - Sparsity pattern
  - Storage format
  - Compiler optimizations
  - Programmer’s optimization effort
Characterizing architectures via CG

Setup

- Target architecture (and compiler)

<table>
<thead>
<tr>
<th>Acron.</th>
<th>Architecture</th>
<th>Total #cores</th>
<th>Frequency (GHz) – Idle power (W)</th>
<th>RAM size, type</th>
<th>Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIL</td>
<td>AMD Opteron 6276 (Interlagos)</td>
<td>8</td>
<td>1.4–167.29, 1.6–167.66 1.8–167.31, 2.1–167.17 2.3–168.90</td>
<td>64GB, DDR3 1.3GHz</td>
<td>icc 12.1.3</td>
</tr>
<tr>
<td>AMC</td>
<td>AMD Opteron 6128 (Magny-Cours)</td>
<td>8</td>
<td>0.8–107.48, 1.0–109.75, 1.2–114.27, 1.5–121.15, 2.0–130.07</td>
<td>48GB, DDR3 1.3GHz</td>
<td>icc 12.1.3</td>
</tr>
<tr>
<td>IAT</td>
<td>Intel Atom S1260</td>
<td>2</td>
<td>0.6–41.94, 0.90–41.93, 1.30–41.97, 1.70–41.95, 2.0–42.01</td>
<td>8GB, DDR3 1.3GHz</td>
<td>icc 12.1.3</td>
</tr>
<tr>
<td>INH</td>
<td>Intel Xeon E5504 (Nehalem)</td>
<td>8</td>
<td>1.60–33.43, 1.73–33.43, 1.87–33.43, 2.00–33.43</td>
<td>32GB, DDR3 1.3GHz</td>
<td>icc 12.1.3</td>
</tr>
<tr>
<td>ISB</td>
<td>Intel E5-2620 (Sandy-Bridge)</td>
<td>6</td>
<td>1.2–113.00, 1.4–112.96, 1.6–112.77, 1.8–112.87, 2.0–112.85</td>
<td>32GB, DDR3 1.3GHz</td>
<td>icc 12.1.3</td>
</tr>
<tr>
<td>A9</td>
<td>ARM Cortex A9</td>
<td>4</td>
<td>0.76–10.0, 1.3–10.1</td>
<td>2GB, DDR3L</td>
<td>gcc 4.6.3</td>
</tr>
<tr>
<td>A15</td>
<td>Exynos5 Octa (ARM Cortex A15 + A7)</td>
<td>4+4</td>
<td>0.25–2.2, 1.6–2.4</td>
<td>2GB, LPDDR3</td>
<td>gcc 4.7</td>
</tr>
<tr>
<td>FER</td>
<td>Intel Xeon E5520 NVIDIA Tesla C2050 (Fermi)</td>
<td>8</td>
<td>1.6–222.0, 2.27–226.0 1.15</td>
<td>24GB, 3GB, GDDR5</td>
<td>gcc 4.4.6</td>
</tr>
<tr>
<td>KEP</td>
<td>Intel Xeon i7-3930K NVIDIA Tesla K20 (Kepler)</td>
<td>6</td>
<td>1.2–106.30, 3.2–106.50 0.7</td>
<td>24GB, 5GB, GDDR5</td>
<td>gcc 4.4.6</td>
</tr>
<tr>
<td>QDR</td>
<td>ARM Cortex A9 NVIDIA Quadro 1000M</td>
<td>2</td>
<td>0.120–11.2, 1.3–12.2 1.4</td>
<td>2GB, DDR3L 2GB, DDR3</td>
<td>gcc 4.6.3</td>
</tr>
<tr>
<td>TIC</td>
<td>Texas Instruments C6678</td>
<td>8</td>
<td>1.0–18.0</td>
<td>512MB, DDR3</td>
<td>cl6x 7.4.1</td>
</tr>
</tbody>
</table>
Characterizing architectures via CG

Setup

- Standard benchmarks

<table>
<thead>
<tr>
<th>Source</th>
<th>Matrix</th>
<th>#nonzeros ((n_z))</th>
<th>Size ((n))</th>
<th>(n_z/n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>UFMC</td>
<td>AUDIKW_1</td>
<td>77,651,847</td>
<td>943,645</td>
<td>82.28</td>
</tr>
<tr>
<td></td>
<td>BMWCRA1</td>
<td>10,641,602</td>
<td>148,770</td>
<td>71.53</td>
</tr>
<tr>
<td></td>
<td>CRANKSEG_2</td>
<td>14,148,858</td>
<td>63,838</td>
<td>221.63</td>
</tr>
<tr>
<td></td>
<td>F1</td>
<td>26,837,113</td>
<td>343,791</td>
<td>78.06</td>
</tr>
<tr>
<td></td>
<td>INLINE_1</td>
<td>38,816,170</td>
<td>503,712</td>
<td>77.06</td>
</tr>
<tr>
<td></td>
<td>LDOOR</td>
<td>42,493,817</td>
<td>952,203</td>
<td>44.62</td>
</tr>
<tr>
<td>Laplace</td>
<td>A100</td>
<td>6,940,000</td>
<td>1,000,000</td>
<td>6.94</td>
</tr>
<tr>
<td></td>
<td>A126</td>
<td>13,907,370</td>
<td>2,000,376</td>
<td>6.94</td>
</tr>
<tr>
<td></td>
<td>A159</td>
<td>27,986,067</td>
<td>4,019,679</td>
<td>6.94</td>
</tr>
<tr>
<td></td>
<td>A200</td>
<td>55,760,000</td>
<td>8,000,000</td>
<td>6.94</td>
</tr>
<tr>
<td></td>
<td>A252</td>
<td>111,640,032</td>
<td>16,003,001</td>
<td>6.94</td>
</tr>
</tbody>
</table>
Characterizing architectures via CG

Setup

- **Storage format**

![Diagram of storage formats for CPU and GPU]
Characterizing architectures via CG

Setup

- Optimization effort:
  - Multicore x86-based: Intel MKL with CSR and BCSR, and CSB library
  - Other multicore: CSR+OpenMP
  - GPUs: ELLPACK & SELL-P, with further optimizations (described in last block)
Characterizing architectures via CG

Setup

- Optimization for run time or energy efficiency?
  - Choose the best combination of frequency-voltage setting, #cores, and storage format to optimize one of them
    - Run time = GFLOPS
    - Energy efficiency = GFLOPS/W
Characterizing architectures via CG

Results
Characterizing architectures via CG Results
Characterizing architectures via CG

Remarks

- GPUs deliver high energy efficiency with outstanding performance for CG
- GFLOPS/W of GPUs can be matched/outperformed by low-power devices
- General-purpose multicore processors provide a reasonable balance between these two extremes

"Unveiling the performance-energy trade-off in iterative linear system solvers for multithreaded processors"
Concurrency and Computation: Practice & Experience, 2015
Outline

- Characterizing architectures via CG
- Energy efficiency of PCG
- Energy saving for multi-core and GPU servers
Energy efficiency of PCG

Compute the preconditioner \( A \rightarrow M \)
Initialize \( x_0, r_0, z_0, d_0, \beta_0, \tau_0 \)
\( k := 0 \)

while \( (\tau_k > \tau_{max}) \)
\[
\begin{align*}
    w_k & := Ad_k \\
    \rho_k & := \beta_k / d_k^T w_k \\
    x_{k+1} & := x_k + \rho_k d_k \\
    r_{k+1} & := r_k - \rho_k w_k \\
    z_{k+1} & := M^{-1} r_{k+1} \\
    \beta_{k+1} & := r_{k+1}^T z_{k+1} \\
    \alpha_k & := \beta_{k+1} / \beta_k \\
    d_{k+1} & := z_{k+1} + \alpha_k d_k \\
    \tau_{k+1} & := \| r_{k+1} \|_2 \\
    k & := k + 1
\end{align*}
\]
endwhile

Iterative PCG solve
(SPMV)
(DOT product)
(APXY)
(APXY)
Preconditioning
(DOT product)
(APXY-like)
(2-norm)
Energy efficiency of PCG

- Incomplete LU Package (http://ilupack.tu-bs.de)
  - Iterative Krylov subspace methods
  - Multilevel ILU preconditioners for general/symmetric/Hermitian positive definite systems
  - Based on inverse ILUs with control over growth of inverse triangular factors
  - Specially competitive for linear systems from 3D PDEs
Energy efficiency of PCG

- Multi-threaded parallelism (real s.p.d. systems)
  - Leverage task parallelism
  - Dynamic scheduling via runtime (OpenMP)
Energy efficiency of PCG

- Run-time in charge of scheduling

"Exploiting thread-level parallelism in the iterative solution of sparse linear systems"
J. I. Aliaga, M. Bollhöfer, A. F. Martín, E. S. Quintana-Ortí
Parallel Computing, 2011
Energy efficiency of PCG
Energy efficiency of PCG

- **Target architectures:**

<table>
<thead>
<tr>
<th>SERVER</th>
<th>CPU</th>
<th>#cores</th>
<th>Freq. (GHz)</th>
<th>Mem (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SANDY</td>
<td>Intel Xeon E5-2620</td>
<td>12</td>
<td>2.0</td>
<td>32 (DDR3)</td>
</tr>
<tr>
<td>HASWELL</td>
<td>Intel Xeon E5-2603v3</td>
<td>12</td>
<td>1.6</td>
<td>32 (DDR3)</td>
</tr>
<tr>
<td>Xeon Phi</td>
<td>Xeon Phi 5110P</td>
<td>60(+1)</td>
<td>1.053</td>
<td>8 (DDR5)</td>
</tr>
<tr>
<td>KEPLER</td>
<td>K40 (GK110B) + Intel i7-4770</td>
<td>2,880</td>
<td>3.40</td>
<td>12 (DDR5)</td>
</tr>
</tbody>
</table>
<pre><code>                                           | 4      |             | 16 (DD3)   |
</code></pre>
Energy efficiency of PCG

- **Architecture tuning:**
  - Exploit task-parallelism on multi-core and Intel Xeon Phi
  - NUMA-aware execution
  - Careful binding of threads/cores on Intel Xeon Phi

- Off-load appropriate kernels to GPU to exploit data-parallelism
## Energy efficiency of PCG

<table>
<thead>
<tr>
<th>Platform</th>
<th>Matrix</th>
<th>Time (s)</th>
<th>GFLOPS</th>
<th>Energy (J)</th>
<th>GFLOPS/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>SANDY</td>
<td>A171</td>
<td>21.12</td>
<td>2.95</td>
<td>2,827.89</td>
<td>0.0221</td>
</tr>
<tr>
<td></td>
<td>A252</td>
<td>101.42</td>
<td>2.74</td>
<td>13,843.17</td>
<td>0.0201</td>
</tr>
<tr>
<td></td>
<td>A318</td>
<td>322.06</td>
<td>2.21</td>
<td>42,827.13</td>
<td>0.0166</td>
</tr>
<tr>
<td>HASWELL</td>
<td>A171</td>
<td>31.89</td>
<td>1.95</td>
<td>3,277.67</td>
<td>0.0193</td>
</tr>
<tr>
<td></td>
<td>A252</td>
<td>154.04</td>
<td>1.80</td>
<td>15,933.05</td>
<td>0.0174</td>
</tr>
<tr>
<td></td>
<td>A318</td>
<td>421.13</td>
<td>1.69</td>
<td>43,419.49</td>
<td>0.0164</td>
</tr>
<tr>
<td>XEON PHI</td>
<td>A171</td>
<td>58.69</td>
<td>1.24</td>
<td>8,032.32</td>
<td>0.0090</td>
</tr>
<tr>
<td>KEPLER</td>
<td>A171</td>
<td>23.09</td>
<td>2.49</td>
<td>2,909.34</td>
<td>0.0198</td>
</tr>
<tr>
<td></td>
<td>A252</td>
<td>83.82</td>
<td>3.16</td>
<td>11,449.81</td>
<td>0.0231</td>
</tr>
</tbody>
</table>
Characterizing architectures via CG

Remarks

- Many-core accelerators generally preferred for their high performance and energy efficiency
- Rapid evolution of recent general-purpose processors with wider SIMD (vector) units and aggressive energy saving mechanisms, blurring part of the energy gap

"Characterizing the Efficiency of Multicore and Manycore Processors for the Solution of Sparse Linear Systems"
J. I. Aliaga, M. Barreda, E. Dufrechou, P. Ezzatti, E. S. Quintana-Ortí
Computer Science – Research and Development, 2015
Outline

- Characterizing architectures via CG
- Energy efficiency of PCG
- Energy saving for multi-core and GPU servers
Energy saving for multi-core servers

- Leveraging P-states during idle periods (DVFS)

Why?
Energy saving for multi-core servers

- Leveraging P-states during idle periods (DVFS)
Energy saving for multi-core servers

- Active polling for work...

![Graph showing power consumption for different thread activities](image)
Energy saving for multi-core servers

- Leveraging C-states during idle periods (C-states)
Energy saving for multi-core servers

Remarks

- Avoid active polling for work from idle threads
- Race-to-idle is more energy-efficient than exploiting P-states even in a memory-bound operation due to large system+static power

"Assessing the impact of the CPU power-saving modes on the task-parallel solution of sparse linear systems"
J. Aliaga, M. Barreda, M. F. Dolz, A. F. Martín, R. Mayo, E. S. Quintana-Ortí
Cluster Computing, 2014
Energy saving for GPU servers

- Leveraging P-states on CPU-GPU platforms?
  - Apply DVFS to the CPU while computation proceeds on the GPU?

- Leveraging C-states on CPU-GPU platforms?
  - What is the CPU doing while computation proceeds on the GPU?
Energy saving for GPU servers
Avoid polling CPU

Initialize $r_0, p_0, x_0, \sigma_0, \tau_0; j := 0$

while $(\tau_j > \tau_{\text{max}})$

$\nu_j := Ap_j$

$\alpha_j := \sigma_j / p_j^T v_j$

$x_{i+1} := x_i + \alpha_i p_i$

endwhile

Loop for iterative CG solver

01. SPMV
02. DOT
03. AXPY

Can we reduce the number of CUDA kernels?
(activation/de-activation of CPU)

$\rho_j := \frac{c_j}{\nu_j}$

$\sigma_{j+1} := \zeta_j$

$p_{j+1} := z_j + \beta_j p_j$

$\tau_{j+1} := \| r_{j+1} \|_2 = \sqrt{\zeta_j}$

$j := j + 1$

06. Scalar op
07. Scalar op
08. XPAY (AXPY-like)
09. Vector 2-norm (in practice, sqrt)
Energy saving for GPU servers
Avoid polling CPU

- Fusion of (i.e., merging) CUDA kernels
  - Separate DOT products into two stages: a+b

...needs modification in the code

"Systematic fusion of CUDA kernels for iterative sparse linear system solvers"
J. I. Aliaga, J. Pérez, E. S. Quintana-Ortí
Euro-Par 2015 (Viena)
Energy saving for GPU servers
Avoiding polling CPU

- Alternative: CUDA “dynamic parallelism” (DP)

“DP is an extension to the CUDA programming model enabling a CUDA kernel to create and synchronize with new work directly on the GPU. […] The ability to create work directly from the GPU can reduce the need to transfer execution control and data between host and device, as launch configuration decisions can now be made at runtime by threads executing on the device”

CUDA Dynamic Parallelism Programming Guide
NVIDIA, August 2012

“Harnessing CUDA DP in the sparse linear system solvers”
J. I. Aliaga, J. Pérez, E. S. Quintana-Ortí
ParCo 2015 (Edinburgh)
Energy saving for GPU servers
Avoiding polling CPU

- Intel i7-3770K, 16GB + NVIDIA Kepler K20c
Energy saving for GPU servers
Avoiding polling CPU

- Intel i7-3770K, 16GB + NVIDIA Kepler K20c
Remarks

- Kernel fusion and DP are orthogonal
- With DP, CPU invokes a single “parent” CUDA kernel to launch the solver on the GPU, and can then be put to sleep
- Necessary to redesign DOT product and AXPY-like operations, into two-stage CUDA kernels, to avoid “nested” invocations to CUDA kernels
Thanks and…

QUESTIONS?