Energy-Aware Linear Algebra

Enrique S. Quintana-Ortí
Concurrency and energy efficiency

- Green500 vs Top500 (June 2013)

<table>
<thead>
<tr>
<th>Rank</th>
<th>Site</th>
<th>Technology</th>
<th>MFLOPS/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/32</td>
<td>Tianhe-2 - National University of Defense Technology</td>
<td>Intel Xeon E5 + Intel Xeon Phi</td>
<td>1.901</td>
</tr>
<tr>
<td>467/1</td>
<td>Eurora - CINECA</td>
<td>Intel Xeon E5 + NVIDIA K20</td>
<td>3.208</td>
</tr>
</tbody>
</table>
Concurrency and energy efficiency

- Green500 vs Top500 (June 2013)

<table>
<thead>
<tr>
<th>Rank</th>
<th>Site</th>
<th>Technology</th>
<th>MFLOPS/W</th>
<th>MW to EXAFLOPS?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/32</td>
<td>Tianhe-2 - National University of Defense Technology</td>
<td>Intel Xeon E5 + Intel Xeon Phi</td>
<td>1.901</td>
<td>408</td>
</tr>
<tr>
<td>467/1</td>
<td>Eurora - CINECA</td>
<td>Intel Xeon E5 + NVIDIA K20</td>
<td>3.208</td>
<td>312</td>
</tr>
</tbody>
</table>

Most powerful reactor under construction in France Flamanville (EDF, 2017 for US $9 billion): 1,630 MWe
Concurrency and energy efficiency

- Green500 vs Top500 (November 2012)

<table>
<thead>
<tr>
<th>Rank</th>
<th>Site</th>
<th>Technology</th>
<th>MFLOPS/W</th>
<th>MW to EXAFLOPS?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/32</td>
<td>Tianhe-2 - National University of Defense Technology</td>
<td>Intel Xeon E5 + Intel Xeon Phi</td>
<td>1.901</td>
<td>408</td>
</tr>
<tr>
<td>467/1</td>
<td>Eurora - CINECA</td>
<td>Intel Xeon E5 + NVIDIA K20</td>
<td>3.208</td>
<td>312</td>
</tr>
</tbody>
</table>

1 MW ≈ $1 Million/year!

Most powerful reactor under construction in France Flamanville (EDF, 2017 for US $9 billion): 1,630 MWe
Concurrency and energy efficiency

- System ranked #1 in Green500
Concurrency and energy efficiency

- System ranked #1 in Green500

Goal: 20MW for 1 EXAFLOP by 2020
Maintaining the improvement rate of last five years (x5) → 40 MW by 2020!!!
Concurrency and energy efficiency

- Reduce energy consumption!
  - Costs over lifetime of an HPC facility often exceed acquisition costs
  - Carbon dioxide is a hazard for health and environment
  - Heat reduces hardware reliability

- Personal view
  - Hardware features some power-saving mechanisms
  - Scientific apps. are in general energy-oblivious
Experimental setup

**AMD**
- 2 AMD Opteron 6128, 48GB
- DVFS per core

<table>
<thead>
<tr>
<th>P-state</th>
<th>$V_{CC_i}$</th>
<th>$f_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>1.23</td>
<td>2.00</td>
</tr>
<tr>
<td>$P_1$</td>
<td>1.17</td>
<td>1.50</td>
</tr>
<tr>
<td>$P_2$</td>
<td>1.12</td>
<td>1.20</td>
</tr>
<tr>
<td>$P_3$</td>
<td>1.09</td>
<td>1.00</td>
</tr>
<tr>
<td>$P_4$</td>
<td>1.06</td>
<td>0.80</td>
</tr>
</tbody>
</table>

**Intel**
- 2 Intel Xeon E5504, 32GB
- DVFS per socket

<table>
<thead>
<tr>
<th>P-state</th>
<th>$V_{CC_i}$</th>
<th>$f_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>1.04</td>
<td>2.00</td>
</tr>
<tr>
<td>$P_1$</td>
<td>0.98</td>
<td>1.73</td>
</tr>
<tr>
<td>$P_2$</td>
<td>0.95</td>
<td>1.60</td>
</tr>
<tr>
<td>$P_3$</td>
<td>1.01</td>
<td>1.87</td>
</tr>
</tbody>
</table>

**C-states:**
- C0: normal operation mode
- C1, C1E: disable core components (L1/L2 caches), clock signal, mem. controller,… increases energy savings at the expense of recovery time
Experimental setup

- National Instruments NI9205+NIcDAQ-9178
- 1,000 Samples/s per channel
Outline

- Modeling power
- Saving power in task-parallel applications
  - ILUPACK for multicore processors
  - CG for hybrid CPU-GPU platforms
- Conclusions
Outline

- Modeling power
- Saving power in task-parallel applications
  - ILUPACK for multicore processors
  - CG for hybrid CPU-GPU platforms
- Conclusions
Modeling Power

\[ P = P_{(S)Y(stem)} + P_{C(PU)} = P_Y + P_{S(static)} + P_{D(dynamic)} \]

- \( P_C \) is the power dissipated by CPU (socket): \( P_S + P_D \)
- \( P_S \) is the static power
- \( P_D \) is dynamic power
- \( P_Y \) is the power of remaining components (e.g., RAM)

Considerations:
- \( P_Y \) and \( P_S \) are constants (though \( P_S \) grows with temperature)
- Hot system
Modeling Power

- System power:

Estimated as *idle* power
Due to off-chip components: e.g., RAM (only mainboard)

\[
P = P^Y + P^S + P^D
\]

\[
P^Y \approx P^I = 80.15 \text{ W}
\]
Modeling Power

- Static power: $P = P^Y + P^S + P^D$

\[ P^T_0(c) = a_0 + b_0 \cdot c = 168.59 + 9.12 \cdot c \text{ W} \]

\[ P^S_0 \approx a_0 - P^Y = 88.44 \text{ W} \]
Modeling Power

- **Dynamic power:**

\[ P = P^Y + P^S + P^D \]

\[ P'T_0(c) = a_0 + b_0 \cdot c = 168.59 + 9.12 \cdot c \text{ W} \]

Busy-wait: \[ P^D_0 \approx b_0 \cdot c = 9.12 \cdot c \text{ W} \]
**Modeling Power**

- **Dynamic power:**

\[ P = P^Y + P^S + P^D \]

![Power dissipated as function of number of active cores](image)

- **Busy-wait:**

\[ P^T_0(c) = a_0 + b_0 c = 168.59 + 9.12 \cdot c \text{ W} \]

- **Busy-wait:**

\[ P^D_0 \approx b_0 c = 9.12 \cdot c \text{ W} \]

An operation more challenging than busy-wait?
Modeling Power

- Task-parallel DLA on multicore and CPU-GPU

Algorithm: $A := \text{CHOL}_\text{LL}(A)$

Partition $A \rightarrow \begin{pmatrix} A_{ll} & A_{lr} \\ A_{rl} & A_{rr} \end{pmatrix}$

where $A_{rr}$ is $0 \times 0$

while $m(A_{rr}) < m(A)$ do

Determine block size $b$

Repartition

$$\begin{pmatrix} A_{ll} & A_{lr} \\ A_{rl} & A_{rr} \end{pmatrix} \rightarrow \begin{pmatrix} A_{ll} & A_{lr} & A_{lr} \\ A_{rl} & A_{rl} & A_{rr} \end{pmatrix}$$

where $A_{ll}$ is $b \times b$

**Variant 1:**

$A_{ll} := \{L \setminus A\}_{ll} = \text{CHOL}_\text{LL}(A_{ll})$

$A_{l1} := A_{ll} \text{TRIL}(A_{ll})^T$ (TRSM)

$A_{l2} := A_{l2} - A_{l1} A_{l1}^T$ (SYRK)

**Variant 2:**

$A_{ll} := A_{ll} \text{TRIL}(A_{ll})^T$ (TRSM)

$A_{ll} := A_{ll} - A_{ll} A_{ll}^T$ (SYRK)

$A_{ll} := \{L \setminus A\}_{ll} = \text{CHOL}_\text{LL}(A_{ll})$

**Variant 3:**

$A_{ll} := A_{ll} - A_{ll} A_{ll}^T$ (SYRK)

$A_{ll} := \{L \setminus A\}_{ll} = \text{CHOL}_\text{LL}(A_{ll})$

$A_{ll} := A_{ll} - A_{ll} A_{ll}^T$ (GEMM)

$A_{ll} := A_{ll} \text{TRIL}(A_{ll})^T$ (TRSM)

Continue with

$$\begin{pmatrix} A_{ll} & A_{lr} \\ A_{rl} & A_{rr} \end{pmatrix} \rightarrow \begin{pmatrix} A_{ll} & A_{lr} & A_{lr} \\ A_{rl} & A_{rl} & A_{rr} \end{pmatrix}$$

endwhile
Modeling Power

- Task-parallel DLA on multicore and CPU-GPU

\[ P_{\text{Chol}}(t) = P^U + P^S + P_{\text{Chol}}^D(t) = P^U + P^S + \sum_{i=1}^{r} \sum_{j=1}^{c} P_{i}^D N_{i,j}(t) \]

<table>
<thead>
<tr>
<th>Task</th>
<th>Block size, b</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>128</td>
</tr>
<tr>
<td>( P_P^D ) (dpotrf)</td>
<td>10.26</td>
</tr>
<tr>
<td>( P_T^D ) (dtrsm)</td>
<td>10.12</td>
</tr>
<tr>
<td>( P_S^D ) (dsyrk)</td>
<td>11.22</td>
</tr>
<tr>
<td>( P_G^D ) (dgemm)</td>
<td>11.98</td>
</tr>
<tr>
<td>( P_B^D ) (busy)</td>
<td>7.62</td>
</tr>
</tbody>
</table>

- Use average Power
- Depends also on #active sockets!
Modeling Power

- Task-parallel DLA on multicore and CPU-GPU
  - Accommodate to memory contention

\[ P_{i}^{D} \rightarrow P_{\{i:j\}}^{D} = \delta_{\{i:j\}} \cdot P_{j}^{M} + (1 - \delta_{\{i:j\}}) \cdot P_{j}^{F} \]

\[ \delta_{\{i:j\}} = \frac{R_{\{i:j\}} - T_{j}(b)}{R_{\{i:j\}}} \]

\[ P_{Op}(t) = P^{Y} + P^{C}(t) \]
\[ = P^{Y} + P^{S} + P_{Op}^{D}(t) \]
\[ = P^{Y} + P^{S} + \sum_{k=1}^{c} \sum_{j=1}^{r} \sum_{i=1}^{n_{j}} P_{\{i:j\}}^{D} \cdot M_{k,\{i:j\}}(t) \]
\[ = P^{Y} + P^{S} + \sum_{k=1}^{c} \sum_{j=1}^{r} \sum_{i=1}^{n_{j}} \left( \delta_{\{i:j\}} \cdot P_{j}^{M} + (1 - \delta_{\{i:j\}}) \cdot P_{j}^{F} \right) \cdot M_{k,\{i:j\}}(t) \]
Modeling Power

- Task-parallel DLA on multicore and CPU-GPU
  - Accommodate memory contention

\[
\begin{pmatrix}
\tilde{\delta}_{j,128} & (1 - \tilde{\delta}_{j,128}) \\
\tilde{\delta}_{j,160} & (1 - \tilde{\delta}_{j,160}) \\
\vdots & \vdots \\
\tilde{\delta}_{j,1024} & (1 - \tilde{\delta}_{j,1024})
\end{pmatrix}
\begin{pmatrix}
P_j^M \\
P_j^F
\end{pmatrix}
= 
\begin{pmatrix}
\tilde{P}_j^{D,128} \\
\tilde{P}_j^{D,160} \\
\vdots \\
\tilde{P}_j^{D,1024}
\end{pmatrix}
\]

<table>
<thead>
<tr>
<th>Task</th>
<th>$P_j^M$</th>
<th>$P_j^F$</th>
<th>$\min_b \delta_{j,b} - \max_b \tilde{\delta}_{j,b}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHOLESKY FACTORIZATION</td>
<td>13.32</td>
<td>18.72</td>
<td>45–86</td>
</tr>
<tr>
<td>TRIANGULAR SOLVE</td>
<td>7.47</td>
<td>15.66</td>
<td>14–28</td>
</tr>
<tr>
<td>SYMMETRIC RANK-b UPDATE</td>
<td>12.83</td>
<td>16.00</td>
<td>15–38</td>
</tr>
<tr>
<td>MATRIX-MATRIX PRODUCT</td>
<td>14.67</td>
<td>15.70</td>
<td>7–15</td>
</tr>
<tr>
<td>LU FACTORIZATION</td>
<td>12.83</td>
<td>17.75</td>
<td>75–95</td>
</tr>
<tr>
<td>TRIANGULAR SOLVE</td>
<td>12.12</td>
<td>19.40</td>
<td>55–80</td>
</tr>
<tr>
<td>2x1 LU FACTORIZATION</td>
<td>12.54</td>
<td>16.54</td>
<td>33–76</td>
</tr>
<tr>
<td>2x1 TRIANGULAR SOLVE</td>
<td>12.53</td>
<td>19.55</td>
<td>81–86</td>
</tr>
<tr>
<td>QR FACTORIZATION</td>
<td>15.30</td>
<td>16.88</td>
<td>62–85</td>
</tr>
<tr>
<td>APPLY ORTH. TRANSF.</td>
<td>12.10</td>
<td>26.98</td>
<td>76–86</td>
</tr>
<tr>
<td>2x1 QR FACTORIZATION</td>
<td>13.91</td>
<td>19.18</td>
<td>65–82</td>
</tr>
<tr>
<td>2x1 APPLY ORTH. TRANSF.</td>
<td>6.84</td>
<td>16.72</td>
<td>16–32</td>
</tr>
<tr>
<td>BUSY WAIT</td>
<td>0</td>
<td>9.21</td>
<td>–</td>
</tr>
</tbody>
</table>
Modeling Power

- Task-parallel DLA on multicore and CPU-GPU
Modeling Power

- Task-parallel DLA on multicore and CPU-GPU
Modeling Power

- Simple, yet accurate:
  - Dense factorizations (Cholesky, LU, QR)
    - Multicore processors
      "Modeling power and energy consumption of dense matrix factorizations on multicore processors"
      P. Alonso, M. F. Dolz, R. Mayo, E. S. Quintana. CCPE 2013
  - CPU-GPU platforms
    "Enhancing performance and energy consumption of runtime schedulers for dense linear algebra"
    P. Alonso, M. F. Dolz, F. D. Igual, R. Mayo, E. S. Quintana. CCPE 2013 (submitted)

- ILUPACK on multicore processors
  "Assessing the impact of the CPU power-saving modes on the task-parallel solution of sparse linear systems"
Outline

- Modeling power
- Saving power in task-parallel appl.
  - ILUPACK for multicore processors
  - CG for hybrid CPU-GPU platforms
- Conclusions
ILUPACK on multicore

- Incomplete LU Package ([http://ilupack.tu-bs.de](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
ILUPACK on multicore
Task parallelism

- Multi-threaded parallelism (real s.p.d. systems)
  - Leverage task parallelism
  - Dynamic scheduling via runtime (OpenMP)
ILUPACK on multicore
Task parallelism

- Run-time in charge of scheduling

ILUPACK on multicore
Experimental setup

- Sparse linear system benchmark
  - Laplacian equation $-\Delta u = f$ in a 3D unit cube $\Omega = [0,1]^3$
  - Linear system $Au = b$ with $A \rightarrow n \times n$, $n = 252^3 \approx 16$ million unknowns and 111 millions of nonzero entries
ILUPACK on multicore
Leveraging P-states (AMD)

<table>
<thead>
<tr>
<th>Platform</th>
<th>P-state, $P_i$</th>
<th>$V_i$</th>
<th>$f_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>WT_AMD</td>
<td>P0</td>
<td>1.23</td>
<td>2.00</td>
</tr>
<tr>
<td></td>
<td>P1</td>
<td>1.17</td>
<td>1.50</td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td>1.12</td>
<td>1.20</td>
</tr>
<tr>
<td></td>
<td>P3</td>
<td>1.09</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>P4</td>
<td>1.06</td>
<td>0.80</td>
</tr>
</tbody>
</table>

- DVFS = P-states (see ACPI standard)
- Moving to a higher P-state results in ↓power
- ↓Power = ↓Energy?
- For a compute-bounded operation, $f_i$ is linear to time$^{-1}$
- In principle, for a memory-bounded operation (ILUPACK), reducing $f_i$ should have a minor impact on performance
ILUPACK on multicore
Leveraging P-states (AMD)

- 1st attempt: Dynamic Static voltage-frequency scaling

<table>
<thead>
<tr>
<th>P-state $P_i$</th>
<th>$T_i$</th>
<th>$P_i^T$</th>
<th>$E_i$</th>
<th>$\Delta T_i$</th>
<th>$\Delta P_i^T$</th>
<th>$\Delta E_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>34.06</td>
<td>282.87</td>
<td>9,634.78</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$P_1$</td>
<td>43.57</td>
<td>235.64</td>
<td>10,267.72</td>
<td>21.88</td>
<td>-16.69</td>
<td>6.53</td>
</tr>
<tr>
<td>$P_2$</td>
<td>54.48</td>
<td>210.86</td>
<td>11,478.79</td>
<td>59.91</td>
<td>-25.45</td>
<td>19.20</td>
</tr>
<tr>
<td>$P_3$</td>
<td>61.58</td>
<td>197.01</td>
<td>12,132.79</td>
<td>80.73</td>
<td>-30.35</td>
<td>25.87</td>
</tr>
<tr>
<td>$P_4$</td>
<td>76.50</td>
<td>186.86</td>
<td>14,295.18</td>
<td>124.47</td>
<td>-33.94</td>
<td>48.28</td>
</tr>
</tbody>
</table>

Why?
ILUPACK on multicore
Leveraging P-states (AMD)

- 1st attempt: Dynamic Static voltage-frequency scaling

<table>
<thead>
<tr>
<th>P-state $P_i$</th>
<th>$V_{cc_i}$</th>
<th>$f_i$</th>
<th>$T_i$</th>
<th>$\Delta T_i$</th>
<th>$BW_i$</th>
<th>$\Delta BW_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>1.23</td>
<td>2.00</td>
<td>34.06</td>
<td></td>
<td>30.29</td>
<td></td>
</tr>
<tr>
<td>$P_1$</td>
<td>1.17</td>
<td>1.50</td>
<td>43.57</td>
<td>21.88</td>
<td>24.63</td>
<td>-18.67</td>
</tr>
<tr>
<td>$P_2$</td>
<td>1.12</td>
<td>1.20</td>
<td>54.48</td>
<td>59.91</td>
<td>20.46</td>
<td>-32.44</td>
</tr>
<tr>
<td>$P_3$</td>
<td>1.09</td>
<td>1.00</td>
<td>61.58</td>
<td>80.73</td>
<td>17.48</td>
<td>-42.30</td>
</tr>
<tr>
<td>$P_4$</td>
<td>1.06</td>
<td>0.80</td>
<td>76.50</td>
<td>124.47</td>
<td>14.00</td>
<td>-53.77</td>
</tr>
</tbody>
</table>

• Combined effect of linear decrease of CPU performance and memory bandwidth!
• Decrease of $P^s_i (P_0 \rightarrow P_2 : -21.47\%)$, decrease of $P^D_i (P_0 \rightarrow P_3 : -60.73\%)$ but $P^y_i$ does not change!
ILUPACK on multicore
Leveraging P-states (AMD)

- 2nd attempt: DVFS during idle periods
ILUPACK on multicore
Leveraging P-states (AMD)

- 2nd attempt: DVFS during idle periods

Why?
ILUPACK on multicore
Leveraging P-states (AMD)

- 2nd attempt: DVFS during idle periods
ILUPACK on multicore

Leveraging P-states (AMD)

- Active polling for work…

![Power for different thread activities graph](image)

- MKL dgemm at 2.00 GHz
- Blocking at 800 MHz
- Polling at 2.00 GHz
- Polling at 800 MHz
ILUPACK on multicore
Leveraging P- and C-states (AMD)

- 3rd attempt: DVFS and idle-wait
ILUPACK on multicore
Leveraging P- and C-states (AMD)

- 3rd attempt: DVFS and idle-wait:
  - Savings of 6.92% of total energy
  - Negligible impact on execution time

- ...but take into account that
  - Idle time: 23.70%
  - Dynamic power: 32.32%
  - Upper bound of savings: 39.32 \cdot 0.2370 = 9.32%
ILUPACK on multicore
Leveraging P-states (Intel)

<table>
<thead>
<tr>
<th>P-state</th>
<th>( VCC_i )</th>
<th>( f_i )</th>
<th>( BW_i )</th>
<th>( \Delta BW_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_0 )</td>
<td>1.04</td>
<td>2.00</td>
<td>12.72</td>
<td>–</td>
</tr>
<tr>
<td>( P_1 )</td>
<td>0.98</td>
<td>1.73</td>
<td>12.61</td>
<td>-0.87</td>
</tr>
<tr>
<td>( P_2 )</td>
<td>0.95</td>
<td>1.60</td>
<td>12.55</td>
<td>-1.34</td>
</tr>
<tr>
<td>( P_3 )</td>
<td>1.01</td>
<td>1.87</td>
<td>12.58</td>
<td>-1.10</td>
</tr>
</tbody>
</table>

**DVFS**

<table>
<thead>
<tr>
<th>( P_i )</th>
<th>( T_i )</th>
<th>( \bar{P}^T_i )</th>
<th>( E_i )</th>
<th>( \Delta T_i )</th>
<th>( \Delta \bar{P}^T_i )</th>
<th>( \Delta E_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ILU</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( P_0 )</td>
<td>56.43</td>
<td>135.17</td>
<td>7,627.97</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>( P_1 )</td>
<td>59.06</td>
<td>127.96</td>
<td>7,557.87</td>
<td>4.67</td>
<td>-5.33</td>
<td>-0.92</td>
</tr>
<tr>
<td>( P_2 )</td>
<td>62.93</td>
<td>121.99</td>
<td>7,676.98</td>
<td>11.52</td>
<td>-9.75</td>
<td>0.64</td>
</tr>
<tr>
<td>( P_3 )</td>
<td>67.05</td>
<td>116.22</td>
<td>7,792.77</td>
<td>18.82</td>
<td>-18.82</td>
<td>2.16</td>
</tr>
<tr>
<td><strong>Solve</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( P_0 )</td>
<td>148.94</td>
<td>155.27</td>
<td>23,123.99</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>( P_1 )</td>
<td>148.52</td>
<td>151.07</td>
<td>22,434.73</td>
<td>-0.28</td>
<td>-2.70</td>
<td>-2.98</td>
</tr>
<tr>
<td>( P_2 )</td>
<td>154.86</td>
<td>145.11</td>
<td>22,469.38</td>
<td>3.97</td>
<td>-6.55</td>
<td>-2.83</td>
</tr>
<tr>
<td>( P_3 )</td>
<td>159.08</td>
<td>138.50</td>
<td>22,033.14</td>
<td>6.81</td>
<td>-10.80</td>
<td>-4.72</td>
</tr>
</tbody>
</table>
ILUPACK on multicore
Leveraging P- and C-states (Intel)

Average reduction: 9.5% for LU and 6.5% for Solve

<table>
<thead>
<tr>
<th></th>
<th>$P_i$</th>
<th>$T_i$</th>
<th>$\tilde{P}^T_i$</th>
<th>$E_i$</th>
<th>$\Delta T_i$</th>
<th>$\Delta \tilde{P}^T_i$</th>
<th>$\Delta E_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ILU</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_0$</td>
<td>56.43</td>
<td>135.17</td>
<td>7,627.97</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>$P_1$</td>
<td>59.06</td>
<td>127.96</td>
<td>7,557.87</td>
<td>4.67</td>
<td>-5.33</td>
<td>-0.92</td>
<td></td>
</tr>
<tr>
<td>$P_2$</td>
<td>62.93</td>
<td>121.99</td>
<td>7,676.98</td>
<td>11.52</td>
<td>-9.75</td>
<td>0.64</td>
<td></td>
</tr>
<tr>
<td>$P_3$</td>
<td>67.05</td>
<td>116.22</td>
<td>7,792.77</td>
<td>18.82</td>
<td>-18.82</td>
<td>2.16</td>
<td></td>
</tr>
<tr>
<td><strong>Solve</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_0$</td>
<td>148.94</td>
<td>155.27</td>
<td>23,123.99</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>$P_1$</td>
<td>148.52</td>
<td>151.07</td>
<td>22,434.73</td>
<td>-0.28</td>
<td>-2.70</td>
<td>-2.98</td>
<td></td>
</tr>
<tr>
<td>$P_2$</td>
<td>154.86</td>
<td>145.11</td>
<td>22,469.38</td>
<td>3.97</td>
<td>-6.55</td>
<td>-2.83</td>
<td></td>
</tr>
<tr>
<td>$P_3$</td>
<td>159.08</td>
<td>138.50</td>
<td>22,033.14</td>
<td>6.81</td>
<td>-10.80</td>
<td>-4.72</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$P_i$</th>
<th>$T_i$</th>
<th>$\tilde{P}^T_i$</th>
<th>$E_i$</th>
<th>$\Delta T_i$</th>
<th>$\Delta \tilde{P}^T_i$</th>
<th>$\Delta E_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ILU</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_0$</td>
<td>55.96</td>
<td>122.83</td>
<td>6,872.82</td>
<td>–</td>
<td>6.34</td>
<td>-5.14</td>
<td>0.87</td>
</tr>
<tr>
<td>$P_1$</td>
<td>59.50</td>
<td>116.52</td>
<td>6,932.86</td>
<td>11.46</td>
<td>-8.47</td>
<td>2.03</td>
<td></td>
</tr>
<tr>
<td>$P_2$</td>
<td>62.37</td>
<td>112.42</td>
<td>7,012.13</td>
<td>19.46</td>
<td>-12.61</td>
<td>4.39</td>
<td></td>
</tr>
<tr>
<td>$P_3$</td>
<td>66.84</td>
<td>107.34</td>
<td>7,714.81</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Solve</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_0$</td>
<td>147.40</td>
<td>146.62</td>
<td>21,594.05</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>$P_1$</td>
<td>148.41</td>
<td>143.18</td>
<td>21,241.34</td>
<td>0.68</td>
<td>-2.35</td>
<td>-1.63</td>
<td></td>
</tr>
<tr>
<td>$P_2$</td>
<td>151.39</td>
<td>138.96</td>
<td>21,037.62</td>
<td>2.71</td>
<td>-5.23</td>
<td>-2.58</td>
<td></td>
</tr>
<tr>
<td>$P_3$</td>
<td>158.28</td>
<td>132.63</td>
<td>20,992.39</td>
<td>7.38</td>
<td>-9.53</td>
<td>-2.79</td>
<td></td>
</tr>
</tbody>
</table>
Outline

- Modeling power
  - ILUPACK for multicore processors

- Saving power in task-parallel appl.
  - ILUPACK for multicore processors
  - CG for hybrid CPU-GPU platforms

- Conclusions
The CG method on CPU-GPU

- 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?
The CG method on CPU-GPU

Experimental setup

- Sandy:
  - Intel i7-3770K, 16GB
  - NVIDIA GeForce GTX480

- Cases from two matrix collections

<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>
The CG method on CPU-GPU
Basic implementation

- CG: Sparse matrix-vector (SpMV) + CUBLAS

```c
while ( ( k < maxiter ) && ( res > epsilon ) ){
    SSpMV <<<Gs,Bs>>> ( n, rowA, colA, valA, d, z );
    tmp = cublasSdot ( n, d, 1, z, 1 );
    rho = beta / tmp;
    gamma = beta;
    cublasSaxpy (n, rho, d, 1, x, 1 );
    cublasSaxpy (n, -rho, z, 1, r, 1 );
    beta = cublasSdot ( n, r, 1, r, 1 );
    alpha = beta / gamma;
    cublasSscal ( n, alpha, d, 1 );
    cublasSaxpy (n, one, r, 1, d, 1 );
    res = sqrt( beta );
    k++;
} // end-while
```
The CG method on CPU-GPU
Basic implementation

- CG: Sparse matrix-vector (SpMV) + CUBLAS

```c
while( ( k < maxiter ) && ( res > epsilon ) ){
    SSpMV <<<Gs,Bs>>>( n, rowA, colA, valA, d, z );
    tmp = cublasSdot( n, d, 1, z, 1 );
    rho = beta / tmp;
    alpha = beta / gamma;
    cublasSscal( n, alpha, d, 1 );
    cublasSaxpy( n, one, r, 1, d, 1 );
    res = sqrt( beta );
    k++;
} // end-while
```

Leveraging P-states:
- Basically all computation performed on the GPU
- Apply static VFS to reduced power in CPU!
The CG method on CPU-GPU
Basic implementation

- CG: Sparse matrix-vector (SpMV) + CUBLAS

```c
while ( ( k < maxiter ) && ( res > epsilon ) ){
    SSpMV <<<Gs,Bs>>> ( n, rowA, colA, valA, d, z );
    tmp = cublasSdot ( n, d, 1, z, 1 );
    rho = beta / tmp;
}
```

Leveraging C-states:
- What is the CPU doing while computation proceeds on the GPU?
- CUDA offers polling (active-wait) vs blocking (idle-wait) operation modes

```c
res = sqrt( beta );
k++;
} // end-while
```
The CG method on CPU-GPU
Basic implementation

- Trading off energy for time: variations of CUDA blocking mode w.r.t. CUDA polling mode
The CG method on CPU-GPU
Basic implementation

- Trading off energy for time: variations of CUDA blocking mode w.r.t. CUDA polling mode

Energy = Time \cdot Power

For AUDIKW_1:
- Time 3.6% ↑
- Power 29.16% ↓
- → Energy 26.6% ↓
The CG method on CPU-GPU Basic implementation

- Trading off energy for time: variations of CUDA blocking mode w.r.t. CUDA polling mode
The CG method on CPU-GPU
Merged implementation

- Can we attain polling performance and blocking energy advantage?
- Requires a reformulation of CG (merge kernels)

```c
while( ( k < maxiter ) && ( res > epsilon ) ){
    sspMV <<<Gs, Bs>>> ( n, rowA, colA, valA, d, z );
    tmp = cublassdot ( n, d, l, z, l );
    rho = beta / tmp;
    gamma = beta;
    cublassaxpy (n, rho, d, l, x, l );
    cublassaxpy (n, -rho, z, l, r, l );
    tmp = cublassdot ( n, r, l, r, l );
    beta = tmp;
    alpha = beta / gamma;
    cublassscal (n, alpha, d, l );
    cublassaxpy (n, one, r, l, d, l );
    res = sqrt( beta );
    k++;
} // end-while
```

```c
while( ( k < maxiter ) && ( res > epsilon ) ){
    scalar_fusion_1 <<<Gs, Bs, Ms>>> ( n, rowA, colA, valA, d, z, beta, rho, gamma, vtmp );
    fusion_2 ( GS, Bs, Ms, n, beta, rho, vtmp );
    fusion_3 <<<Gs, Bs, Ms>>> ( n, rho, d, x, z, r, vtmp );
    fusion_4 ( GS, Bs, Ms, n, vtmp, vtmp2 );
    fusion_5 <<<Gs, Bs>>> ( n, beta, gamma, alpha, d, r, vtmp );
    cudaMemcpy( &res, beta, sizeof(float), cudaMemcpyDeviceToHost );
    res = sqrt( beta );
    k ++;
} // end-while
```
The CG method on CPU-GPU Merged implementation

- Time vs. CPU energy

Maintain performance of polling…

...while leveraging energy-efficiency of C-states+idle-wait
Performance and energy consumption
Summary

“Do nothing, efficiently…” (V. Pallipadi, A. Belay)

or

“Doing nothing well” (D. E. Culler)
Acknowledgments

A. F. Martín

H. Anzt

J. I. Aliaga, M. Barreda, M. F. Dolz, R. Mayo, J. Pérez, E. S. Quintana-Ortí

P. Alonso
Acknowledgments

- EU FP7 318793 Project "EXA2GREEN. Energy-Aware Sustainable Computing on Future Technology - Paving the Road to Exascale Computing"

- Project CICYT TIN2011-23283 "PA-HPC. Power-Aware High Performance Computing"