TX: Algorithmic Energy Saving for Distributed Dense Matrix Factorizations

Li Tan and Zizhong Chen

University of California, Riverside

ScalA'14, co-located with SC'14, New Orleans, LA, USA
November 17, 2014
Power Management in HPC via DVFS

> Power and energy consumption of high performance computing is a growing severity → operating costs and system reliability.

> Dynamic Voltage and Frequency Scaling (DVFS)
  > voltage/frequency ↓ → power ↓ → energy efficiency
  > Strategically switch processors to low-power states when the peak processor performance is unnecessary
  > Slack: communication delay, load imbalance, etc.
Effectiveness of DVFS Approaches

 Basics of DVFS
  A runtime technique that is able to switch *operating frequency* and *supply voltage* of power-scalable hardware components (CPU, GPU, memory, etc.) to different *levels* per *workload characteristics* to energy

 In this Work → CPU DVFS
  Various handy DVFS interfaces: CPUFreq
  CPU energy costs dominate the total system energy consumption of a HPC system without accelerators
Two Classic Energy Saving Solutions
Limitations of Existing Solutions

- **OS Level Solutions**
  - Working aside running applications and thus requiring no *application-specific knowledge* and no source mod.
  - High *generality* but less *speciality*
  - Making *online* energy saving decisions via dynamic monitoring and analysis on *workload characteristics*
  - Online workload prediction can be *inaccurate*

- **Parallel Cholesky, LU, and QR Factorizations**
  - Linear Algebra lib w/ *variable* execution characteristics
  - The remaining unfinished global matrix *shrinks* in runs
Limitations of Existing Solutions (Cont.)

› OS Level Solutions: *Effectiveness*
  › Rely on workload prediction to calculate the slack
    › A simple assumption that task behavior is *identical* every time a task is executed in an *iterative* workload
  › Can be defective for parallel matrix factorizations
    › Length variation of iterations of the core loop makes the prediction *inaccurate* ➔ invalidate potential energy savings

› OS Level Solutions: *Completeness*
  › Work when tasks are *being executed*
  › Untapped energy savings for durations when not all tasks are launched and finished due to *dependencies*
Our Approach

- Library Level *Race-to-halt* DVFS Scheduling
  - Task Dependency Set (TDS) analysis based on *algorithmic characteristics* ➔ trading off *partial generality* for *higher energy efficiency*
  - Critical Path (CP) detection and CP-aware slack analysis/reclamation are avoided
  - The idea is intended for any *task-parallel* models where *data flow analysis* can be applied
  - The use of TDS analysis as a compiler technique allows possible extension to a *general* compiler-based approach based on *static analysis*
Cholesky, LU, and QR Factorizations

- Solving Systems of Linear Equations $A\mathbf{x} = \mathbf{b}$
  - Cholesky: symmetric positive definite matrices
  - LU/QR: any general $M \times N$ matrices
  - Solve $LL^T\mathbf{x} = \mathbf{b}$, $PLU\mathbf{x} = \mathbf{b}$, $QR\mathbf{x} = \mathbf{b}$ easily

Stepwise LU Factorization without Pivoting
Task Dependency Set and Critical Path

- Two TDS for Each Task $t$: $TDS_{in}(t)$ and $TDS_{out}(t)$
  - TDS is statically generated using *algorithmic characteristics* of parallel Cholesky/LU/QR factorization
  - TDS is dynamically maintained using *data dependency information* among parallel tasks

- CP: a particular task trace with the total slack of 0
  - CP pinpoints potential energy savings in terms of slack
  - CP can be effective to identify *computation slack* $\rightarrow$ the essence of *possible* extra energy savings
  - CP can be generated by many means: Here we produce CP via *static TDS analysis*
## CP-aware and TX Approaches

<table>
<thead>
<tr>
<th>Task</th>
<th>CP-aware (Slack Reclamation)</th>
<th>TX (Race-to-halt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comp. Tasks on the CP</td>
<td>highest</td>
<td>highest</td>
</tr>
<tr>
<td>Comp. Tasks off the CP</td>
<td>reduce frequency to dilate task into slack</td>
<td>run-highest idle-lowest</td>
</tr>
<tr>
<td>Comm. Tasks on/off the CP</td>
<td>lowest</td>
<td>lowest</td>
</tr>
</tbody>
</table>

### Timing of DVFS Scheduling

- CP-aware: respects exec. info. of previous iterations
- TX: respects dependency info. of parallel tasks
Algorithm 3  DVFS Scheduling Algorithm Using CP

\textbf{DVFS\_CP}(CritPath, task, FreqSet)
1: if (task \in CritPath \mid TDS_{out}(task) \neq \emptyset) then
2: SetFreq(f_h)
3: else
4: \quad slack \leftarrow \text{GetSlack}(task)
5: \quad if (slack > 0) then
6: \quad \quad f_{opt} \leftarrow \text{GetOptFreq}(task, slack)
7: \quad \quad if (f_i \leq f_{opt} \leq f_h) then
8: \quad \quad \quad if (f_{opt} \notin FreqSet) then
9: \quad \quad \quad \quad SetFreq([f_{opt}, [f_{opt}], ratio])
10: \quad \quad \quad else SetFreq(f_{opt})
11: \quad \quad else if (f_{opt} < f_i) then
12: \quad \quad \quad SetFreq(f_i)
13: \quad end if
14: end if

Algorithm 4  DVFS Scheduling Algorithm Using TX

\textbf{DVFS\_TX}(task, CurFreq)
1: while (TDS_{in}(task) \neq \emptyset) do
2: \quad if (CurFreq \neq f_i) then
3: \quad \quad SetFreq(f_i)
4: \quad \quad if (Recv(DoneFlag, t_1)) then
5: \quad \quad \quad delete(TDS_{in}(task), t_1)
6: \quad \quad end if
7: \quad end while
8: SetFreq(f_h)
9: if (IsFinished(task)) then
10: \quad \quad foreach t_2 \in TDS_{out}(task) do
11: \quad \quad \quad Send(DoneFlag, t_2)
12: \quad \quad \quad SetFreq(f_i)
13: \quad \quad end if
Implementation

- State-of-the-art OS Level Solutions and Our TX
  - Fermata: OS level, only handles comm. tasks
  - Adagio: OS level, handles comp. tasks based on CP-aware workload prediction. Fermata is incorporated
  - CPU-Speed: OS level, based on CPU utilization
  - SC_lib: Library level, only handles comm. tasks
  - TX: Library level, handle comp. tasks based on race-to-halt TDS analysis. SC_lib is incorporated
  - CP_theo (power only): Library level, theoretical CP-aware slack reclamation of Adagio
# Hardware Configuration

<table>
<thead>
<tr>
<th>Cluster</th>
<th>HPCL (Energy + Perf.)</th>
<th>ARC (Power + Perf.)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System Size</strong></td>
<td>8</td>
<td>108</td>
</tr>
<tr>
<td><strong>Processor</strong></td>
<td>2 x Quad-core</td>
<td>2 x 8-core</td>
</tr>
<tr>
<td></td>
<td>AMD Opteron 2380</td>
<td>AMD Opteron 6128</td>
</tr>
<tr>
<td><strong>CPU Frequency</strong></td>
<td>0.8, 1.3, 1.8, 2.5 GHz</td>
<td>0.8, 1.0, 1.2, 1.5, 2.0 GHz</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>8 GB RAM</td>
<td>64 GB RAM</td>
</tr>
<tr>
<td><strong>Network</strong></td>
<td>1GB/s Ethernet</td>
<td>40GB/s Infiniband</td>
</tr>
<tr>
<td><strong>OS</strong></td>
<td>CentOS 6.2, 64-bit</td>
<td>CentOS 5.7, 64-bit</td>
</tr>
<tr>
<td></td>
<td>Linux kernel 2.6.32</td>
<td>Linux kernel 2.6.32</td>
</tr>
<tr>
<td><strong>Power Meter</strong></td>
<td>PowerPack</td>
<td>Watts up? PRO</td>
</tr>
</tbody>
</table>
Power Savings

Matrix Size: 160000 x 160000, Power Costs of Three Nodes
Energy and Performance Efficiency

- 25600 x 25600 Cholesky Factorization with Different Energy Saving Solutions
- 25600 x 25600 LU Factorization with Different Energy Saving Solutions
- 25600 x 25600 QR Factorization with Different Energy Saving Solutions

- Cholesky Factorizations with Different Global Matrix Sizes and Energy Saving Solutions
- LU Factorizations with Different Global Matrix Sizes and Energy Saving Solutions
- QR Factorizations with Different Global Matrix Sizes and Energy Saving Solutions
Conclusions

- Library Level Race-to-halt DVFS Scheduling
  - TDS analysis based on algorithmic characteristics
  - Parallel Cholesky, LU, and QR factorizations

- Compared to Application Level Solutions
  - Restrict source modification and recompilation at library level, allowing replacement of the energy efficient libraries at link time ➔ *partial loss of generality*

- Compared to OS Level Solutions
  - Circumvent the defective workload prediction, and save extra energy from possible load imbalance ➔ *higher energy efficiency*