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

UCR
UNIVERSITY OF CALIFORNIA, RIVERSIDE

Power Management in HPC via DVFS

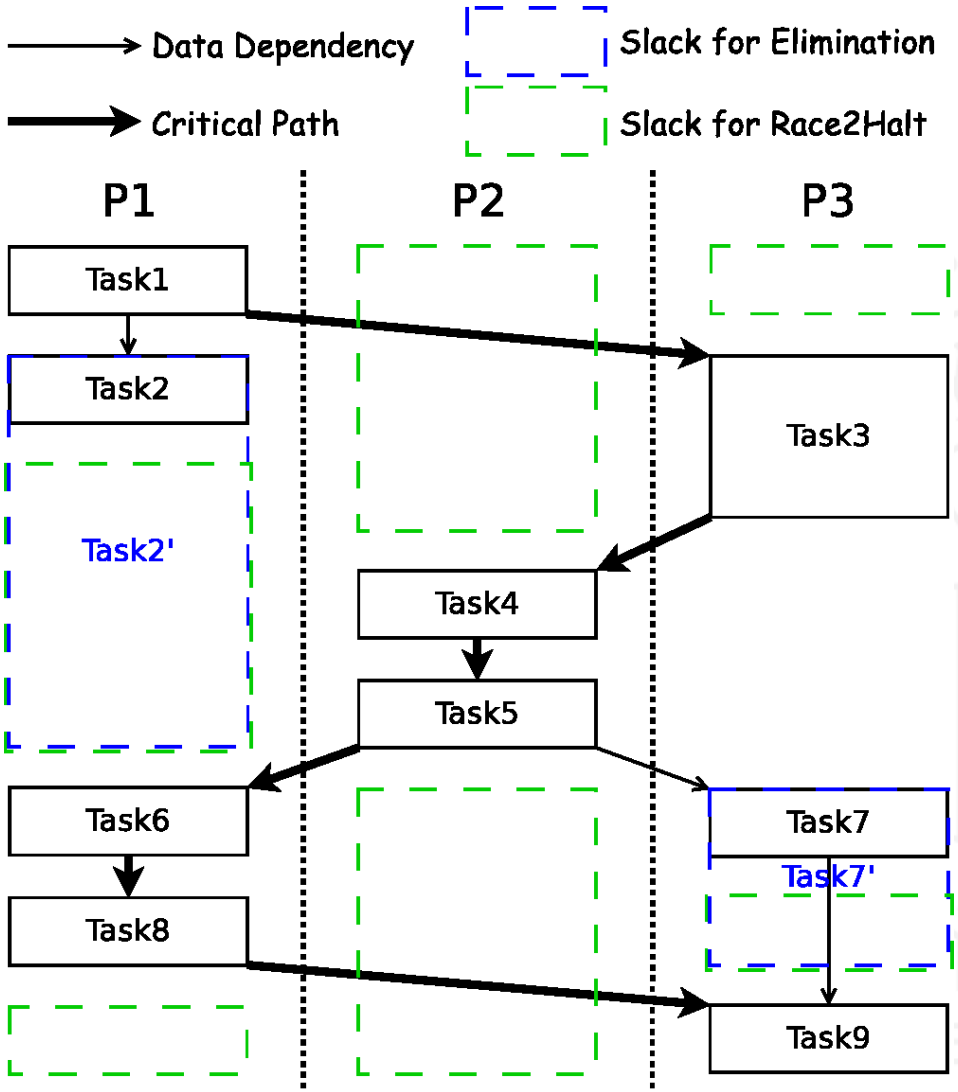
- Power and energy consumption of high performance computing is a growing severity → *operating costs* and *system reliability*.

- 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.)

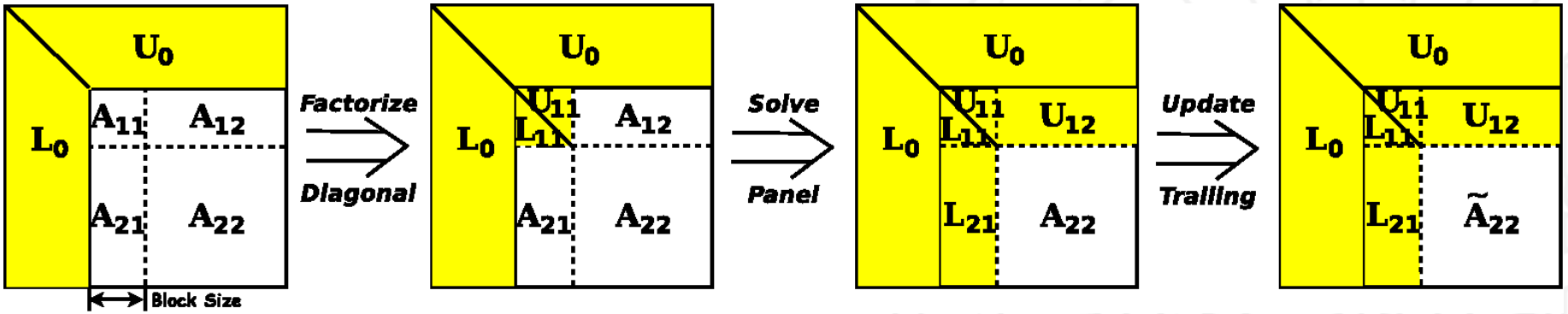
- OS Level Solutions: *Effectiveness*
  - Rely on workload prediction to calculate the slack
    - A simple assumption that task behavior is *identical* every time a task is executed in an *iterative* workload
  - Can be defective for parallel matrix factorizations
    - Length variation of iterations 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

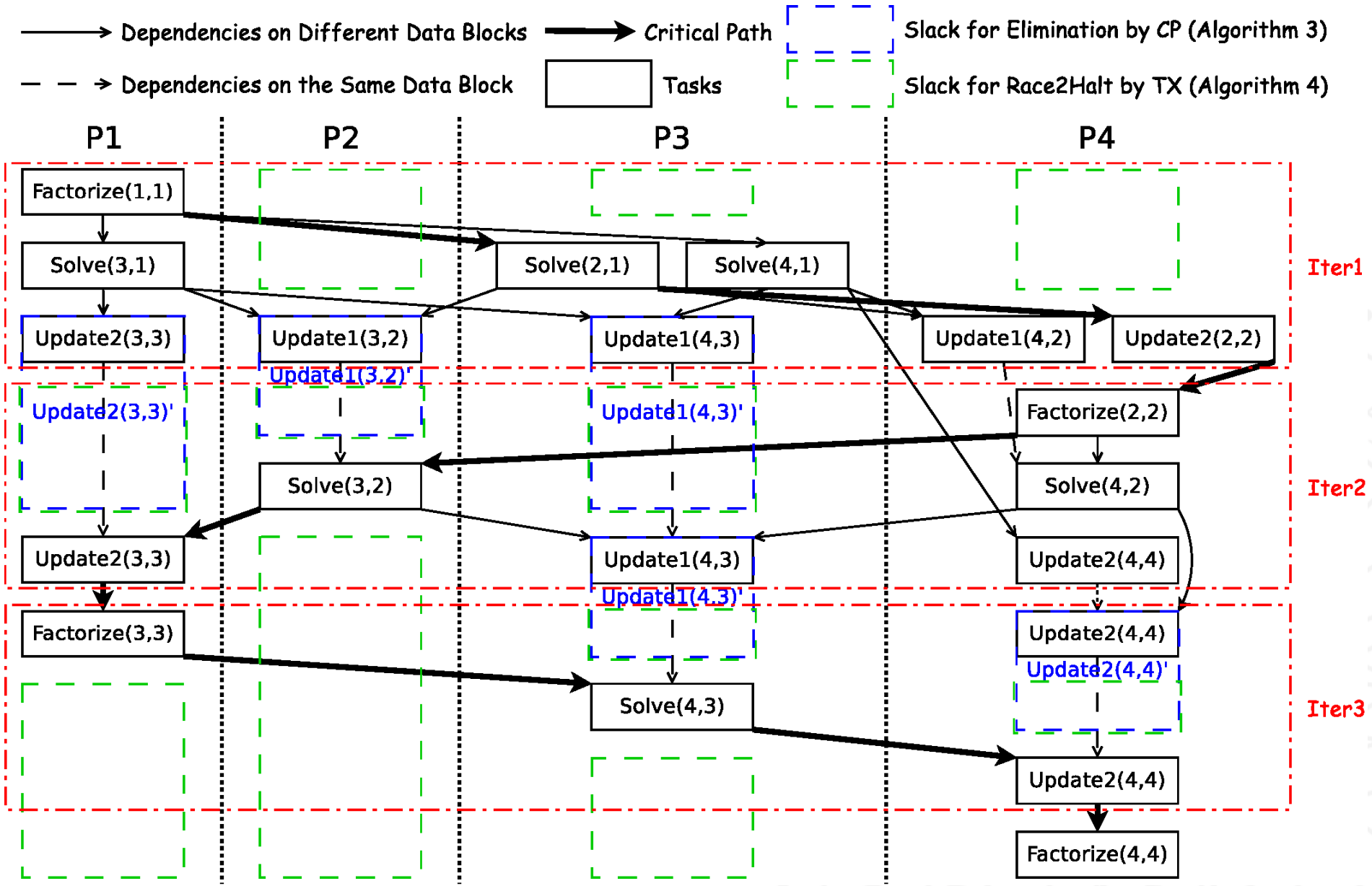
- Solving Systems of Linear Equations $Ax = b$
  - Cholesky: symmetric positive definite matrices
  - LU/QR: any general M x N matrices
  - Solve $LL^Tx = b$, $PLUx = b$, $QRx = b$ easily



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* → the essence of *possible* extra energy savings
  - CP can be generated by many means: Here we produce CP via *static TDS analysis*


Dependencies on Different Data Blocks
Critical Path
Slack for Elimination by CP (Algorithm 3)
Dependencies on the Same Data Block
Tasks
Slack for Race2Halt by TX (Algorithm 4)
P1
P2
P3
P4
Factorize(1,1)
Solve(3,1)
Solve(2,1)
Solve(4,1)
Update2(3,3)
Update1(3,2)
Update1(4,3)
Update1(4,2)
Update2(2,2)
Update1(3,2)'
Update2(3,3)'
Update1(4,3)'
Factorize(2,2)
Solve(3,2)
Solve(4,2)
Update2(3,3)
Update1(4,3)
Update2(4,4)
Update1(4,3)'
Factorize(3,3)
Update2(4,4)
Update2(4,4)'
Solve(4,3)
Update2(4,4)
Factorize(4,4)
Iter1
Iter2
Iter3

CP-aware and TX Approaches

Task	CP-aware (Slack Reclamation)	TX (Race-to-halt)
Comp. Tasks on the CP	highest	highest
Comp. Tasks off the CP	reduce frequency to dilate task into slack	run-highest idle-lowest
Comm. Tasks on/off the CP	lowest	lowest

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

CP-aware and TX Approaches (Cont.)

Algorithm 3 *DVFS Scheduling Algorithm Using CP*

```
DVFS_CP(CritPath, task, FreqSet)
 1: if (task ∈ CritPath || TDS_out(task) != ∅) then
 2:    SetFreq(f_h)
 3: else
 4:    slack ← GetSlack(task)
 5:    if (slack > 0) then
 6:       f_opt ← GetOptFreq(task, slack)
 7:    if (f_l ≤ f_opt ≤ f_h) then
 8:       if (f_opt ∉ FreqSet) then
 9:          SetFreq(⌊f_opt⌋, ⌈f_opt⌉, ratio)
10:       else SetFreq(f_opt)
11:    else if (f_opt < f_l) then
12:       SetFreq(f_l)
13: end if
```

Algorithm 4 *DVFS Scheduling Algorithm Using TX*

```
DVFS_TX(task, CurFreq)
 1: while (TDS_in(task) != ∅) do
 2:    if (CurFreq != f_l) then
 3:       SetFreq(f_l)
 4:    if (Recv(DoneFlag, t_1)) then
 5:       delete(TDS_in(task), t_1)
 6: end while
 7: SetFreq(f_h)
 8: if (IsFinished(task)) then
 9:    foreach t_2 ∈ TDS_out(task) do
10:       Send(DoneFlag, t_2)
11:    SetFreq(f_l)
12: end if
```

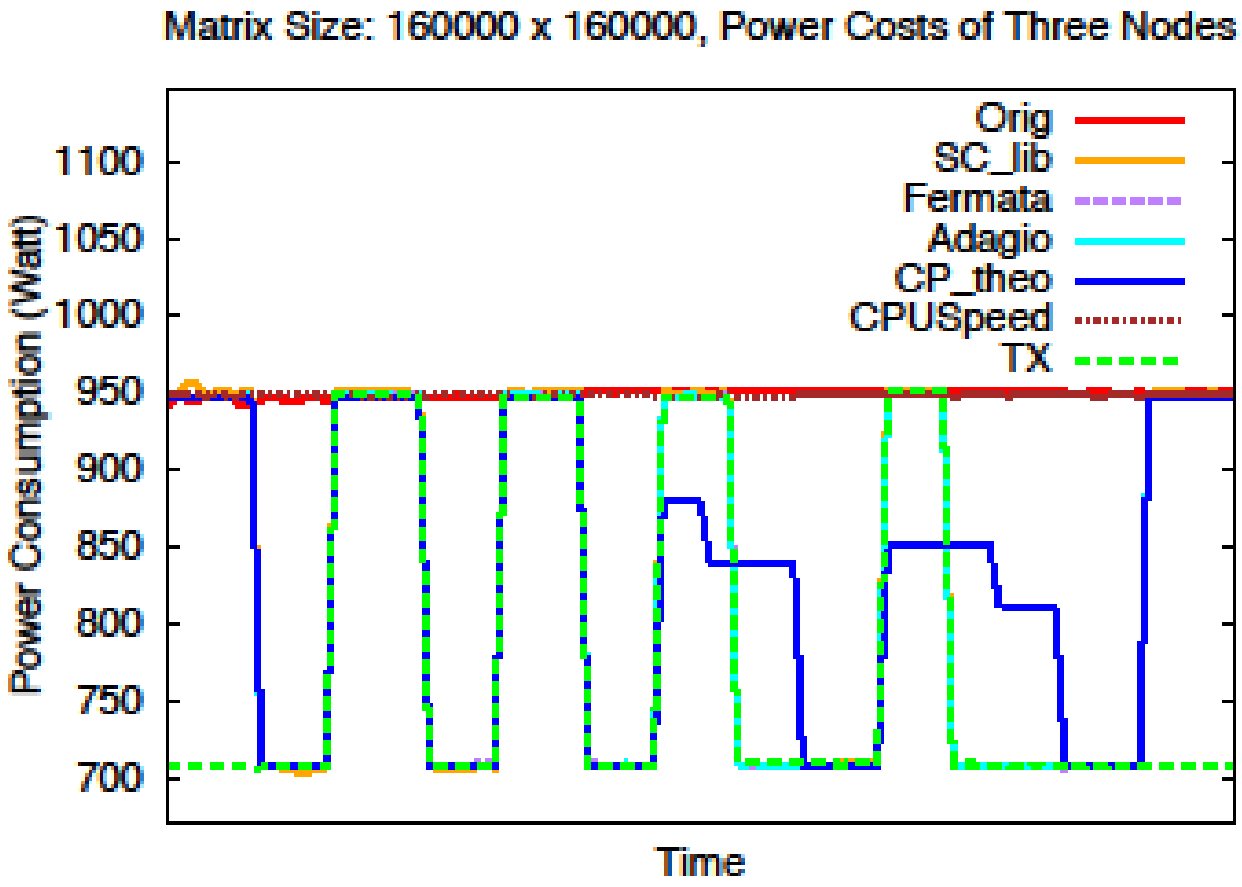
Implementation

- 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
  - *CPUSpeed* : 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*

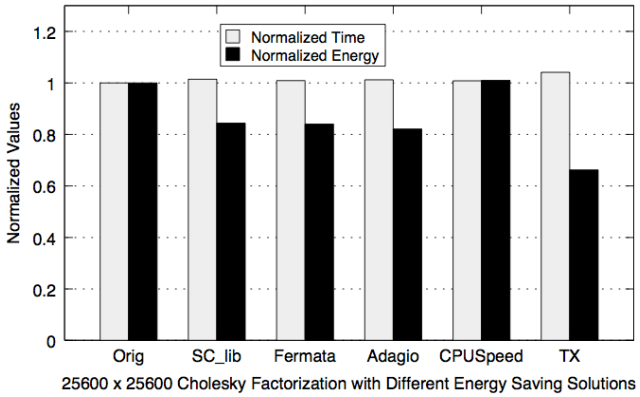
Hardware Configuration

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

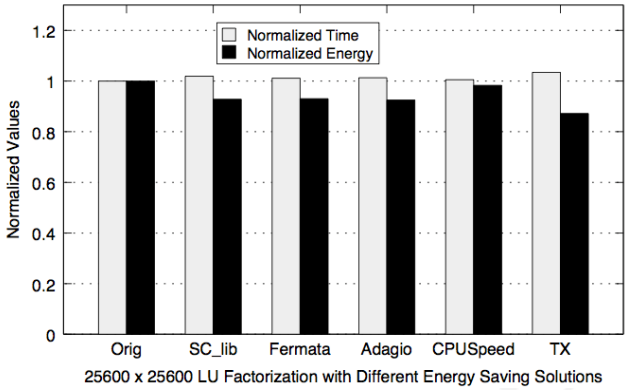
Power Savings



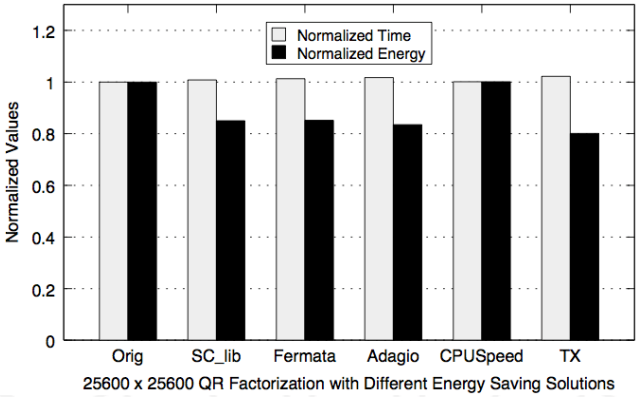
Energy and Performance Efficiency



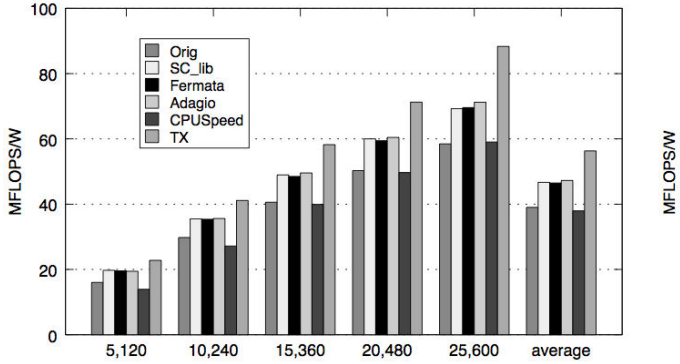
25600 x 25600 Cholesky Factorization with Different Energy Saving Solutions



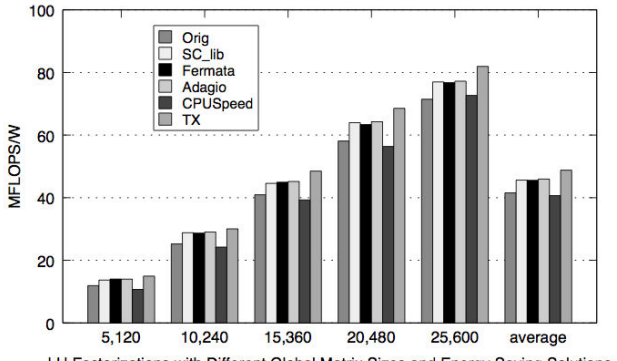
25600 x 25600 LU Factorization with Different Energy Saving Solutions



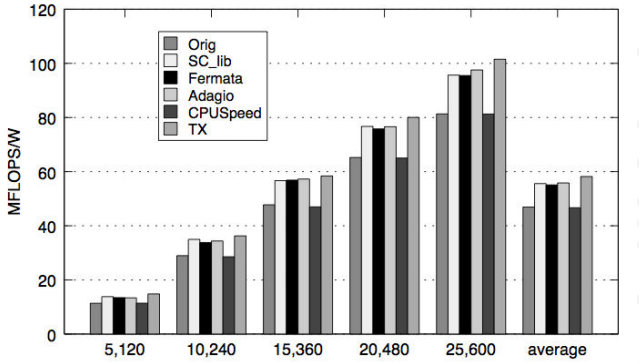
25600 x 25600 QR Factorization with Different Energy Saving Solutions



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*