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

Initialize $r_0, p_0, x_0, \sigma_0, \tau_0; j := 0$
while $(\tau_j > \tau_{\max})$
$v_j := A p_j$
$\alpha_j := \sigma_j / p_j^T v_j$
$x_{j+1} := x_j + \alpha_j p_j$
$r_{j+1} :=$
$\zeta_j := r_{j+1}^T r_{j+1}$
$\beta_j := \zeta_j / \sigma_j$
$\sigma_{j+1} := \zeta_j$
$p_{j+1} := z_j + \beta_j p_j$
$\tau_{j+1} := \| r_{j+1} \|_2 = \sqrt{\zeta_j}$
$j := j + 1$
endwhile

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

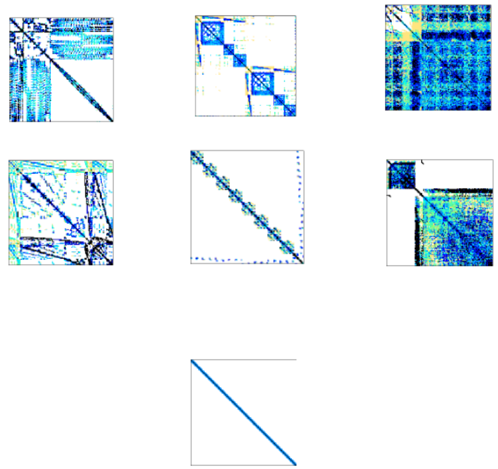
- Target architecture (and compiler)

Acron.	Architecture	Total #cores	Frequency (GHz) – Idle power (W)	RAM size, type	Compiler
AIL	AMD Opteron 6276 (Interlagos)	8	1.4–167.29, 1.6–167.66 1.8–167.31, 2.1–167.17 2.3–168.90	64GB, DDR3 1.3GHz	icc 12.1.3
AMC	AMD Opteron 6128 (Magny-Cours)	8	0.8–107.48, 1.0–109.75, 1.2–114.27, 1.5–121.15, 2.0–130.07	48GB, DDR3 1.3GHz	icc 12.1.3
IAT	Intel Atom S1260	2	0.6–41.94, 0.90–41.93, 1.30–41.97, 1.70–41.95 2.0–42.01	8GB, DDR3 1.3GHz	icc 12.1.3
INH	Intel Xeon E5504 (Nehalem)	8	1.60–33.43, 1,73–33.43, 1.87–33.43, 2.00–33.43	32GB, DDR3 800MHz	icc 12.1.3
ISB	Intel E5-2620 (Sandy-Bridge)	6	1.2–113.00, 1.4–112.96, 1.6–112.77, 1.8–112.87, 2.0–112.85	32GB, DDR3 1.3GHz	icc 12.1.3
A9	ARM Cortex A9	4	0.76–10.0, 1.3–10.1	2GB, DDR3L	gcc 4.6.3
A15	Exynos5 Octa (ARM Cortex A15 + A7)	4+4	0.25–2.2, 1.6–2.4	2GB, LPDDR3	gcc 4.7
FER	Intel Xeon E5520 NVIDIA Tesla C2050 (Fermi)	8 448	1.6–222.0, 2.27–226.0 1.15	24GB, 3GB, GDDR5	gcc 4.4.6 nvcc 5.5
KEP	Intel Xeon i7-3930K NVIDIA Tesla K20 (Kepler)	6 2,496	1.2–106.30, 3.2–106.50 0.7	24GB, 5GB, GDDR5	gcc 4.4.6 nvcc 5.5
QDR	ARM Cortex A9 NVIDIA Quadro 1000M	4 96	0.120–11.2, 1.3–12.2 1.4	2GB, DDR3L 2GB, DDR3	gcc 4.6.3 nvcc 5.5
TIC	Texas Instruments C6678	8	1.0–18.0	512MB, DDR3	cl6x 7.4.1

Characterizing architectures via CG
Setup

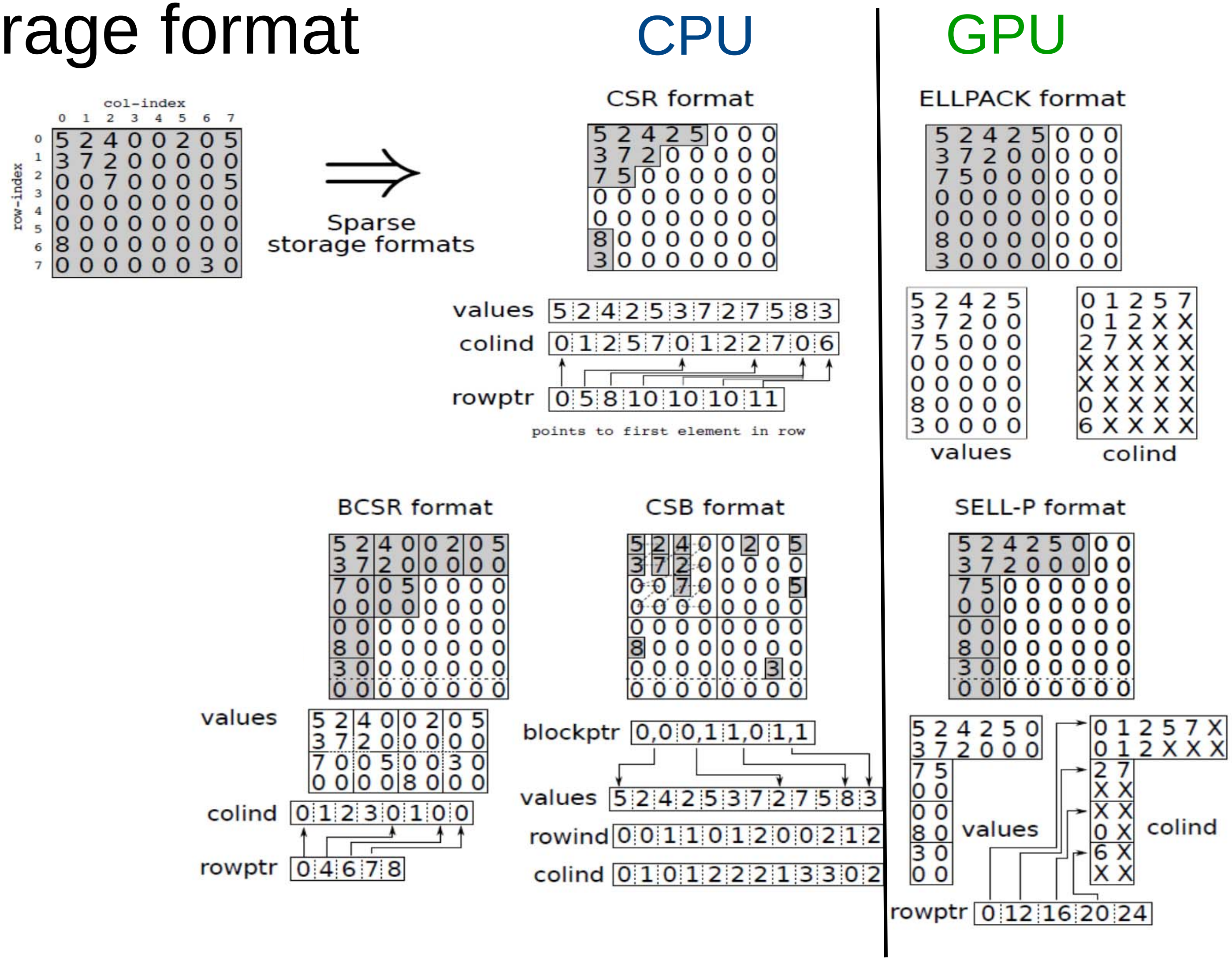
- Standard benchmarks

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



Characterizing architectures via CG Setup

- Storage format



Characterizing architectures via CG Setup

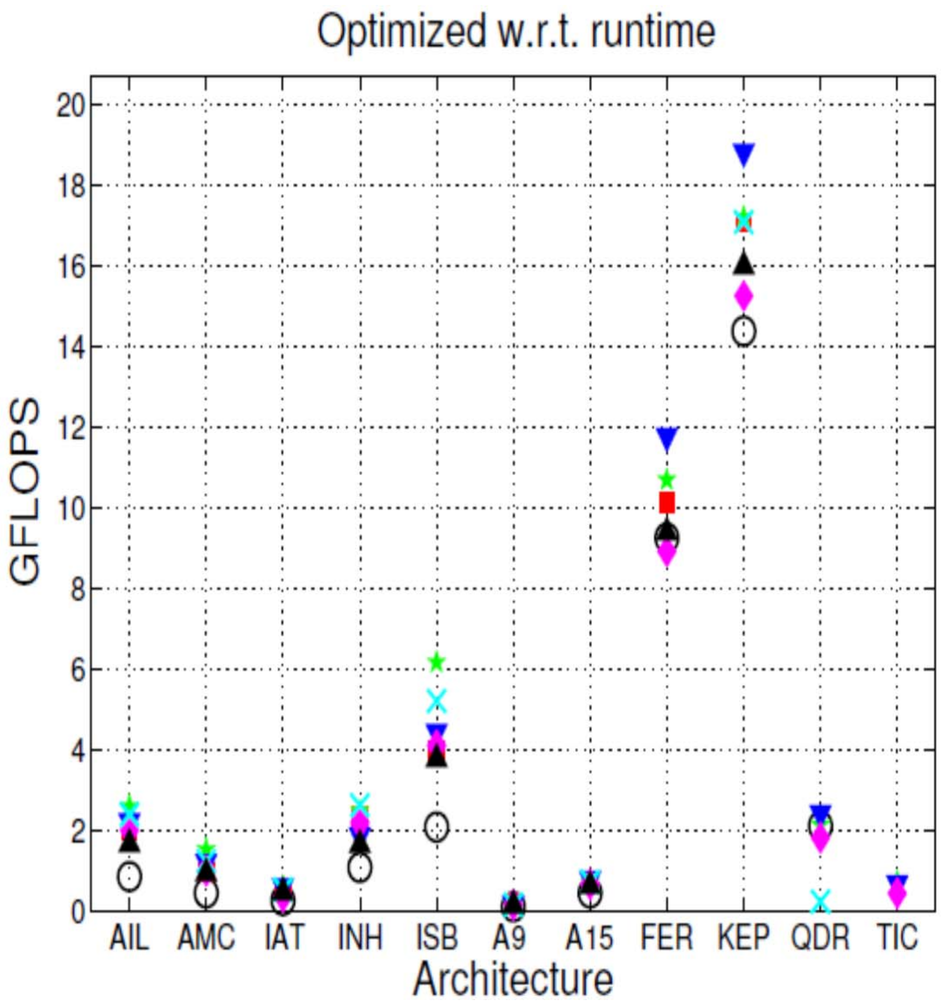
- Optimization effort:
  - Multicore x86-based: Intel MKL with CSR and BCSR, and CSB library
  - Other multicore: CSR+OpenMP
  - GPUs: ELLPACK & SELL-P, with further optimizations (described in last block)

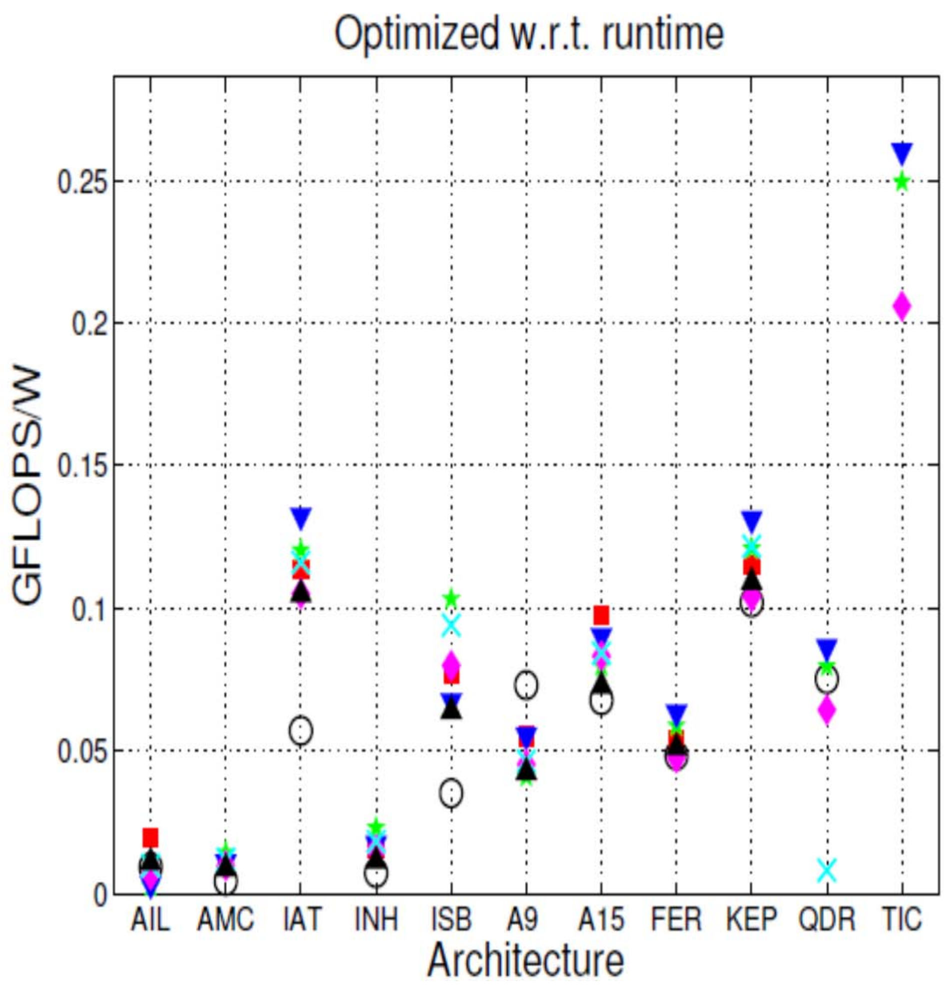
Characterizing architectures via CG Setup

- Optimization for run time or energy efficiency?
  - Choose the best combination of frequency-voltage setting, #cores, and storage format to optimize one of them

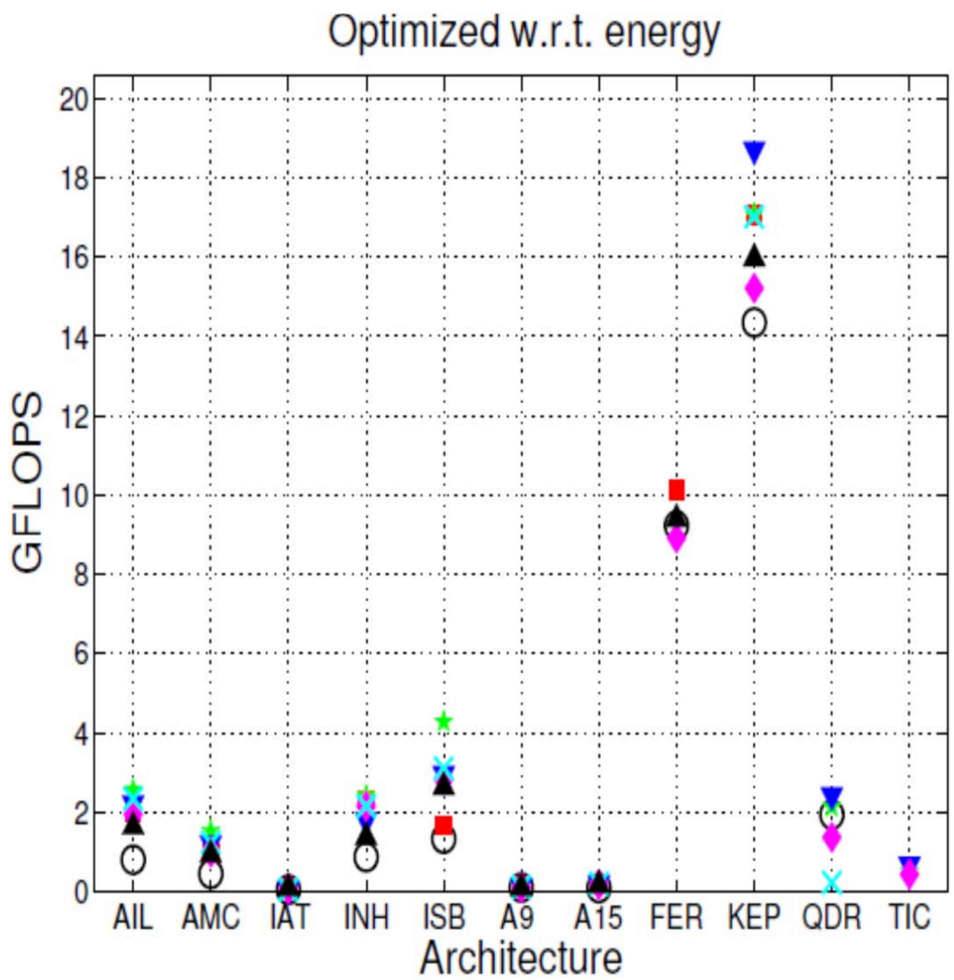
  - Run time = GFLOPS
  - Energy efficiency = GFLOPS/W

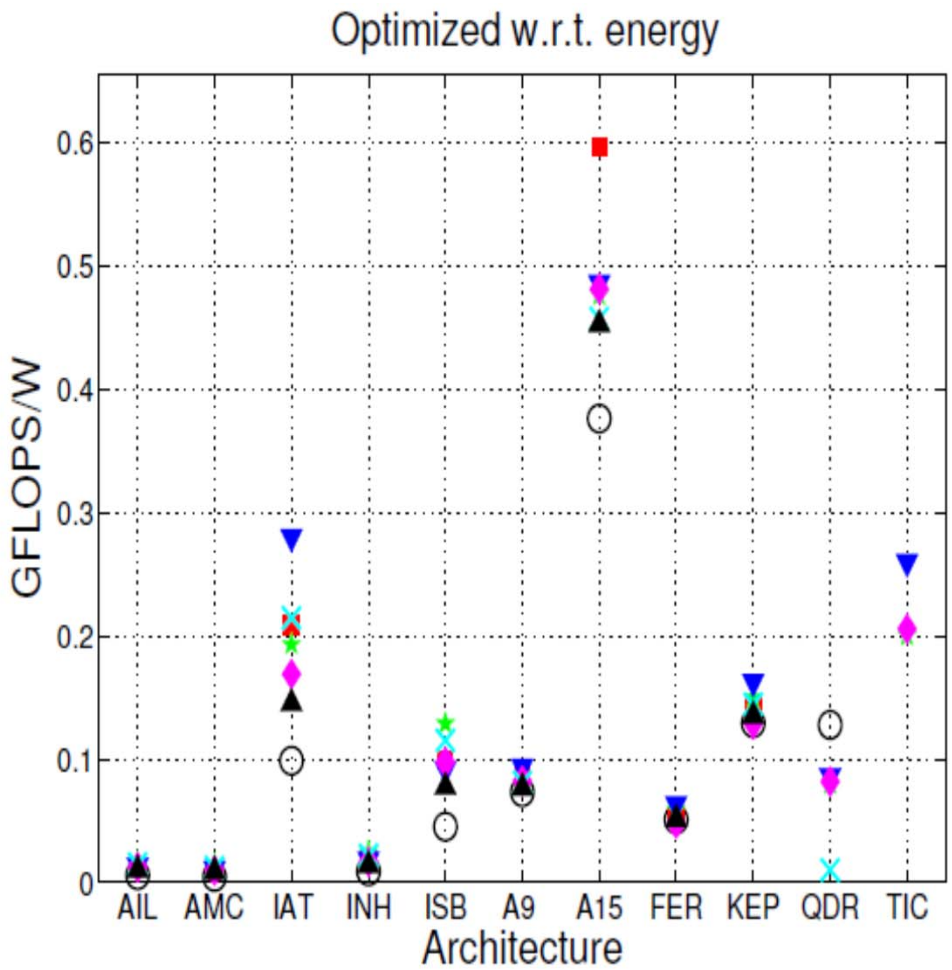
Characterizing architectures via CG Results





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"
J. I. Aliaga, H. Anzt, M. Castillo, J. Fernández, G. León, J. Pérez, E. S. Quintana-Ortí
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

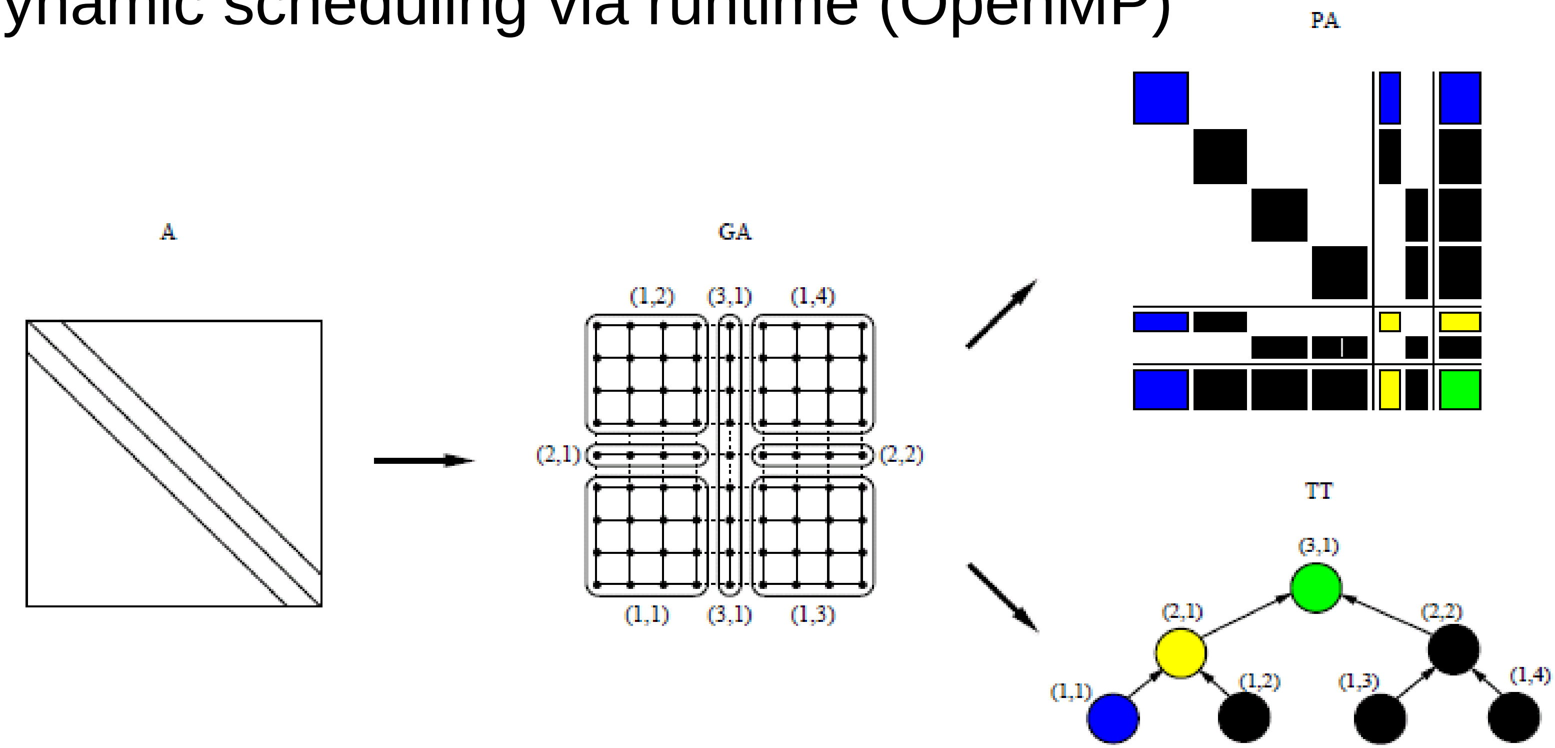
Compute the preconditioner $A \rightarrow M$	
Initialize $x_0, r_0, z_0, d_0, \beta_0, \tau_0$	
$k := 0$	
while $(\tau_k > \tau_{\max})$	**Iterative PCG solve**
$w_k := Ad_k$	(SPMV)
$\rho_k := \beta_k / d_k^T w_k$	(DOT product)
$x_{k+1} := x_k + \rho_k d_k$	(AXPY)
$r_{k+1} := r_k - \rho_k w_k$	(AXPY)
$z_{k+1} := M^{-1} r_{k+1}$	Preconditioning
$\beta_{k+1} := r_{k+1}^T z_{k+1}$	(DOT product)
$\alpha_k := \beta_{k+1} / \beta_k$	
$d_{k+1} := z_{k+1} + \alpha_k d_k$	(AXPY-like)
$\tau_{k+1} := \parallel r_{k+1} \parallel_2$	(2-norm)
$k := k + 1$	
endwhile	

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

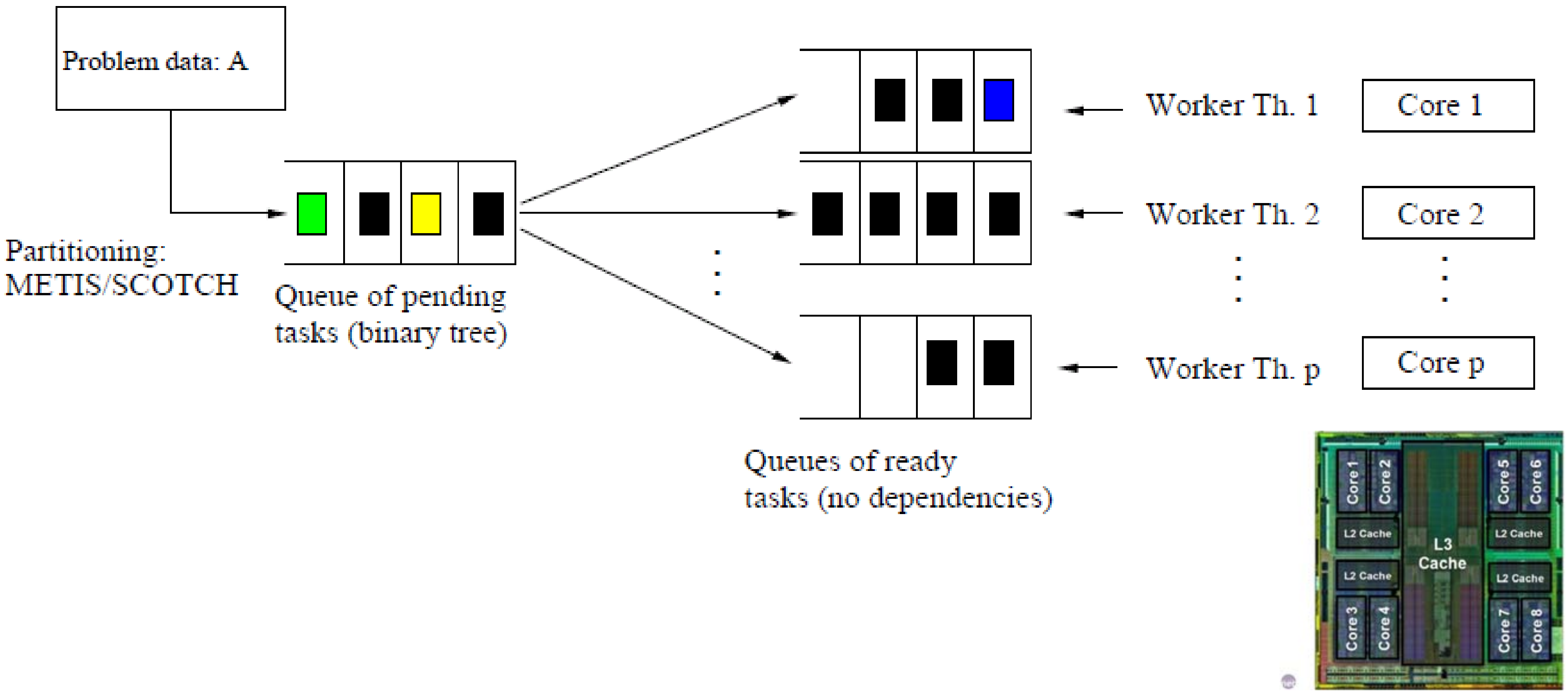
Energy efficiency of PCG

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



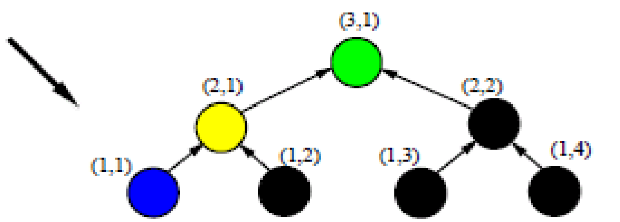
Energy efficiency of PCG

- 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:

SERVER	CPU	#cores	Freq. (GHz)	Mem (GB)
SANDY	Intel Xeon E5-2620	12	2.0	32 (DDR3)
HASWELL	Intel Xeon E5-2603v3	12	1.6	32 (DDR3)
Xeon Phi	Xeon Phi 5110P	60(+1)	1.053	8 (DDR5)
KEPLER	K40 (GK110B) + Intel i7-4770	2,880 + 4	3.40	12 (DDR5) 16 (DD3)

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

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

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

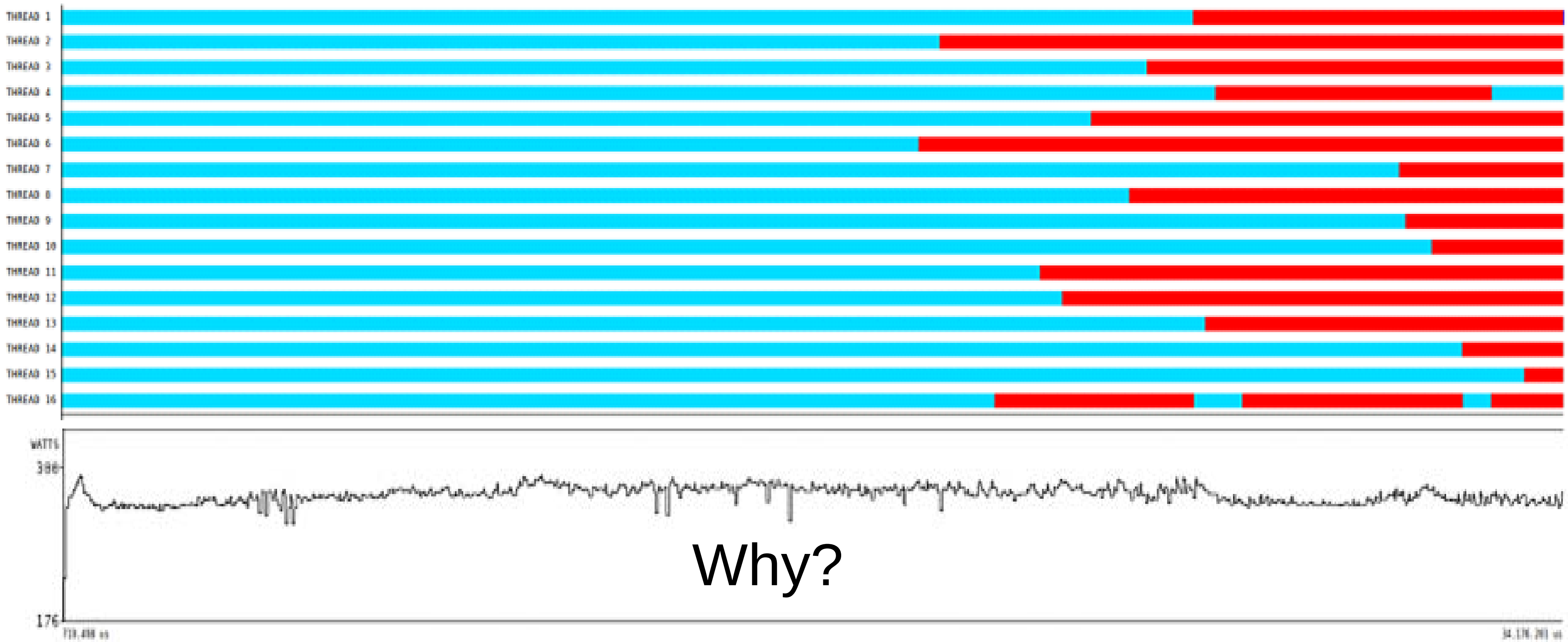
"Characterizing the Eciency of Multicore and Manycore Processors for the Solution of Sparse Linear Systems"
J. I. Aliaga, M. Barreda, E. Dufrechou, P. Ezzatti, E. S. Quintana-Ortí
Computer Science – Research and Development, 2015

Outline

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

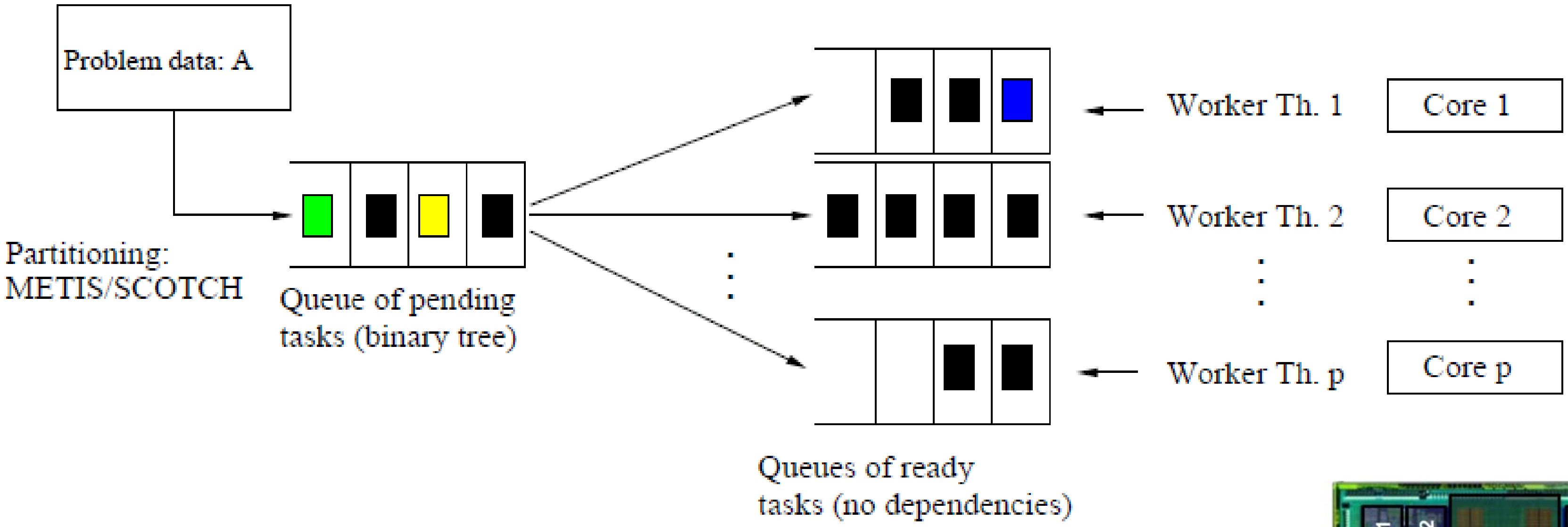
Energy saving for multi-core servers

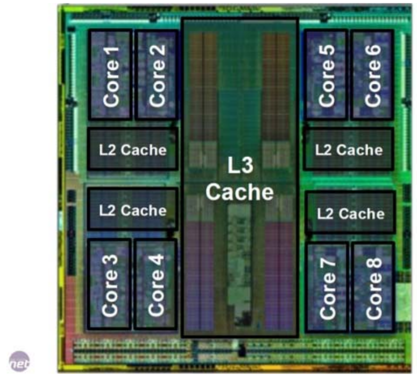
- Leveraging P-states during idle periods (DVFS)



Energy saving for multi-core servers

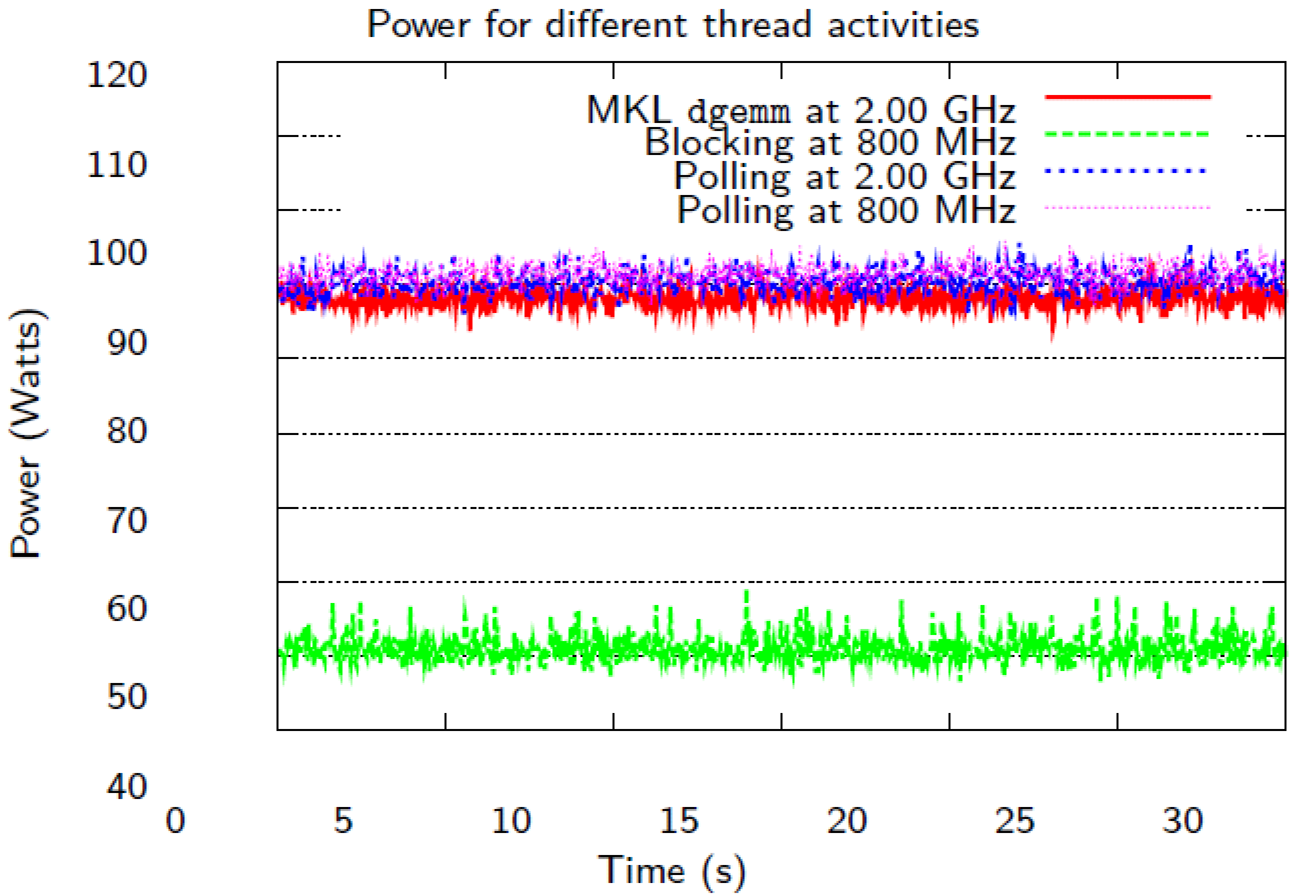
- Leveraging P-states during idle periods (DVFS)





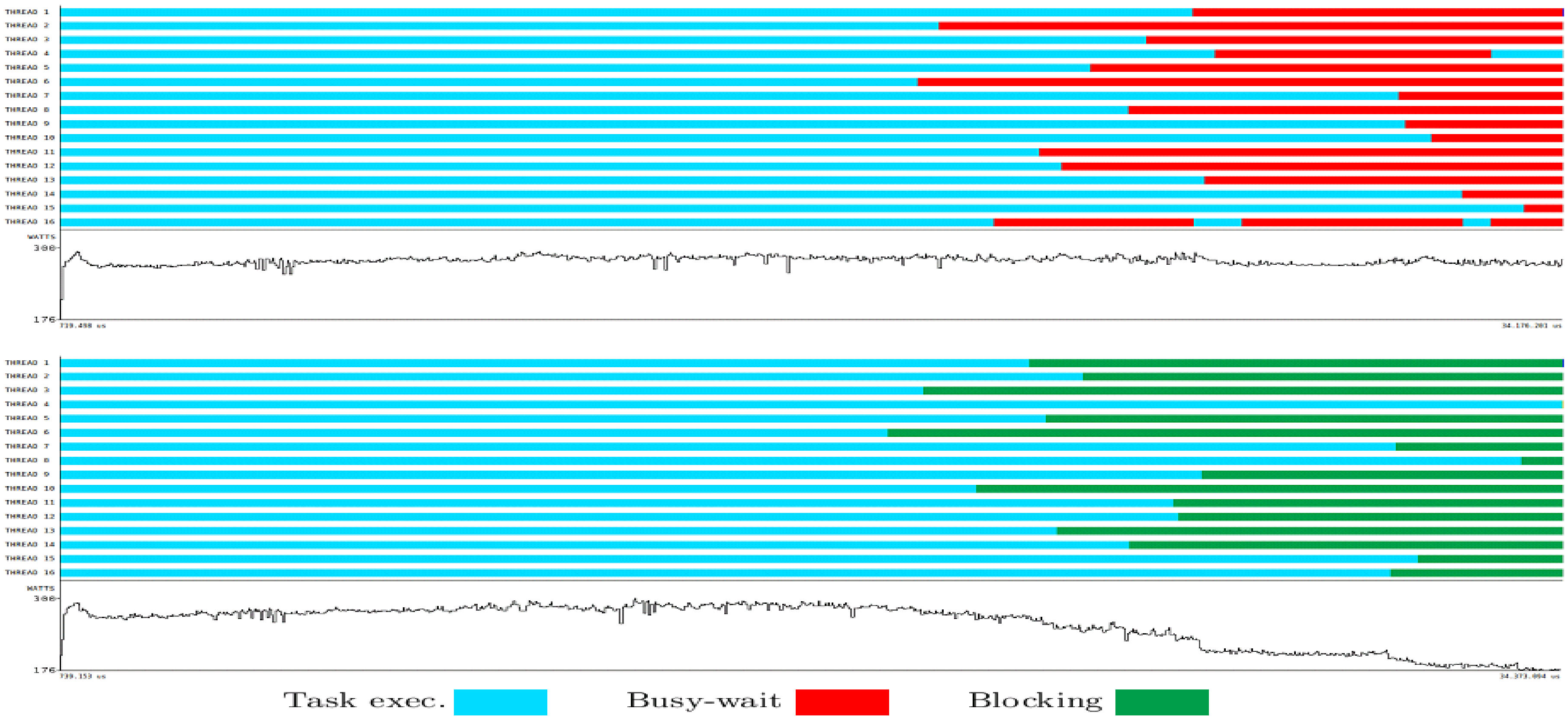
Energy saving for multi-core servers

- Active polling for work…



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?

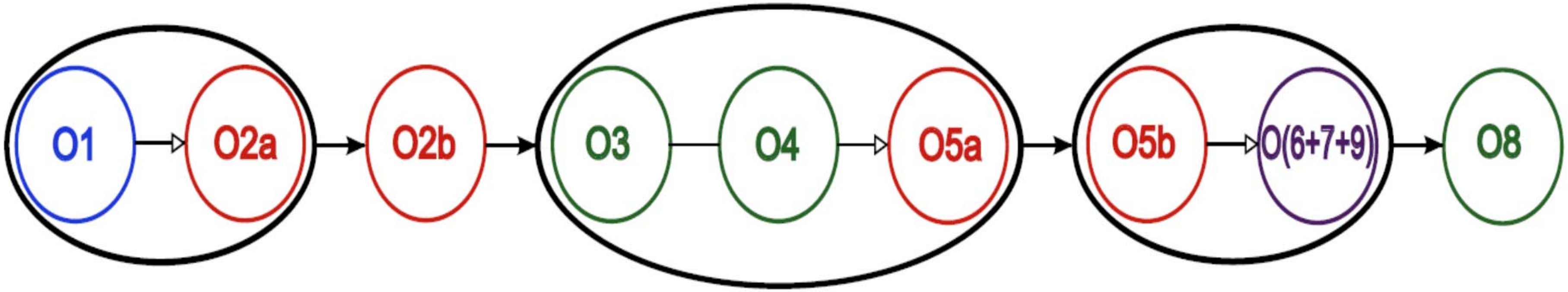
Energy saving for GPU servers
Avoid polling CPU

Initialize $r_0, p_0, x_0, \sigma_0, \tau_0; j := 0$	
while $(\tau_j > \tau_{\max})$	Loop for iterative CG solver
$v_j := A p_j$	O1. SpMV
$\alpha_j := \sigma_j / p_j^T v_j$	O2. DOT
$x_{j+1} := x_j + \alpha_j p_j$	O3. AXPY
$\sigma_{j+1} := \zeta_j$	O7. Scalar op
$p_{j+1} := z_j + \beta_j p_j$	O8. XPAY (AXPY-like)
$\tau_{j+1} := \parallel r_{j+1} \parallel_2 = \sqrt{\zeta_j}$	O9. Vector 2-norm (in practice, sqrt)
$j := j + 1$	
endwhile	

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

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

UNIVERSITAT JAUME·I

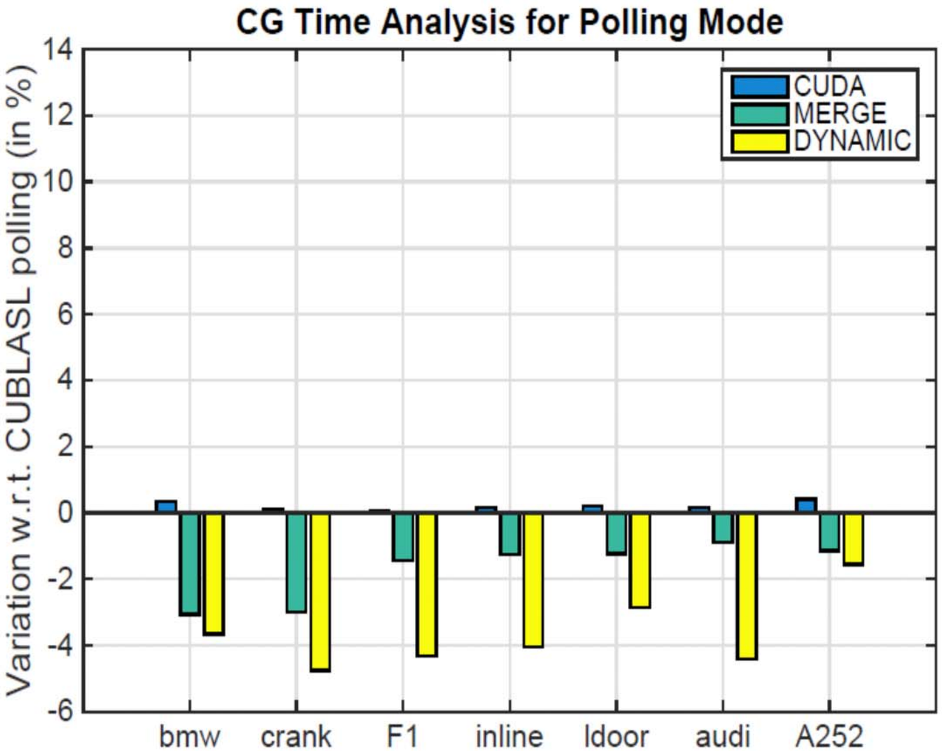
- Alternative: CUDA “dynamic parallelism” (DP)

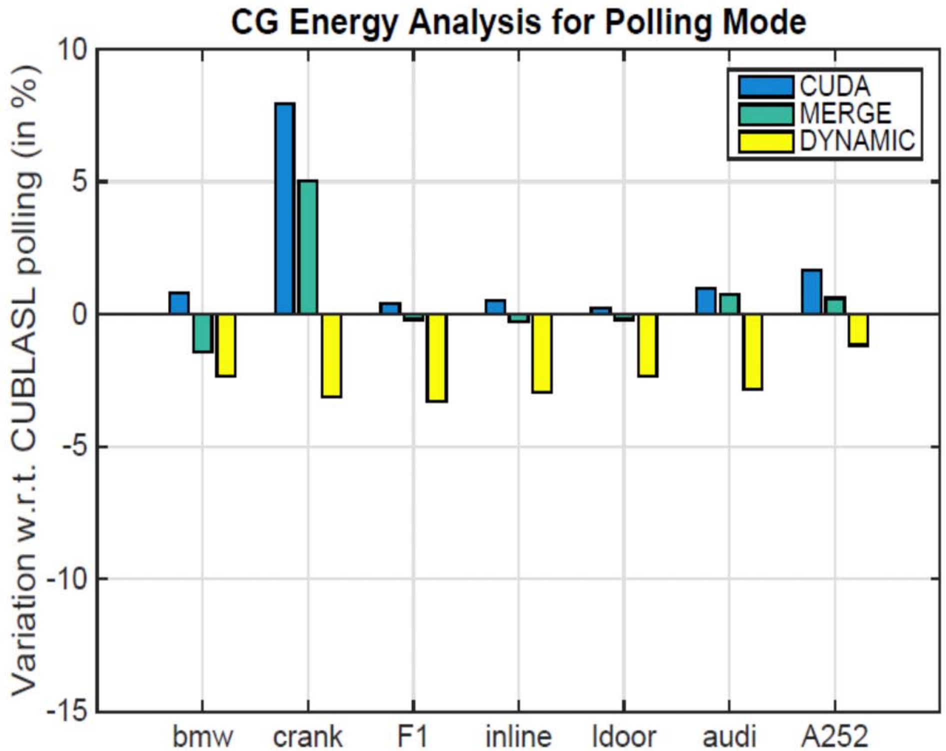
> “DP is an extension to the CUDA programming model enabling a CUDA kernel to create and synchronize with new work directly on the GPU. […] The ability to create work directly from the GPU can reduce the need to transfer execution control and data between host and device, as launch configuration decisions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

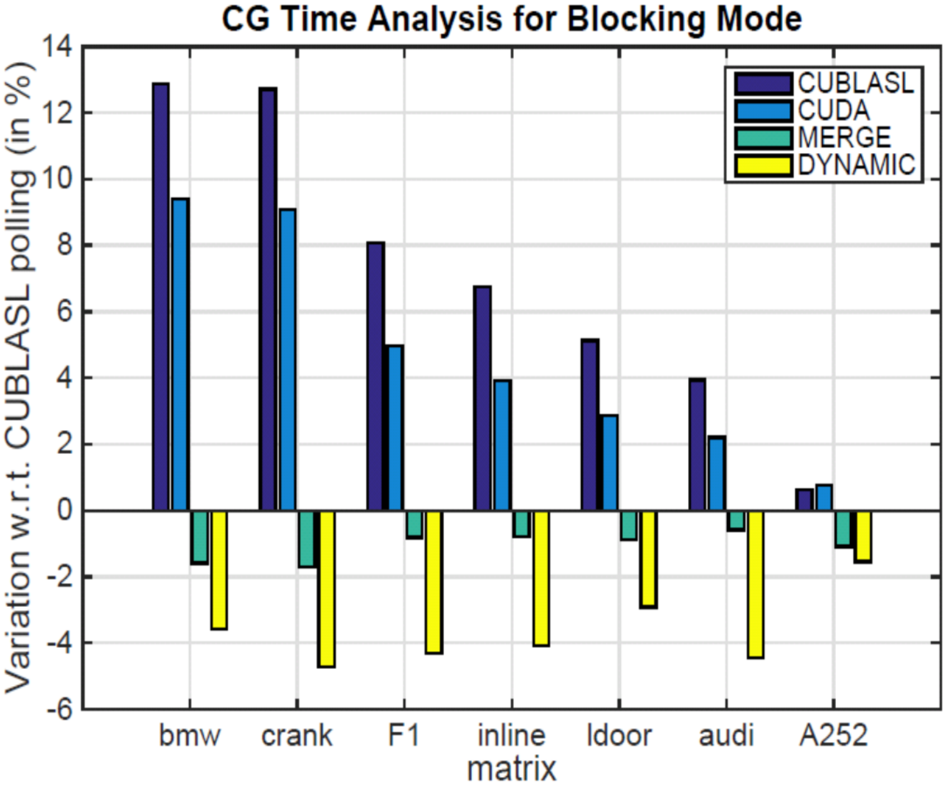
- Intel i7-3770K, 16GB + NVIDIA Kepler K20c

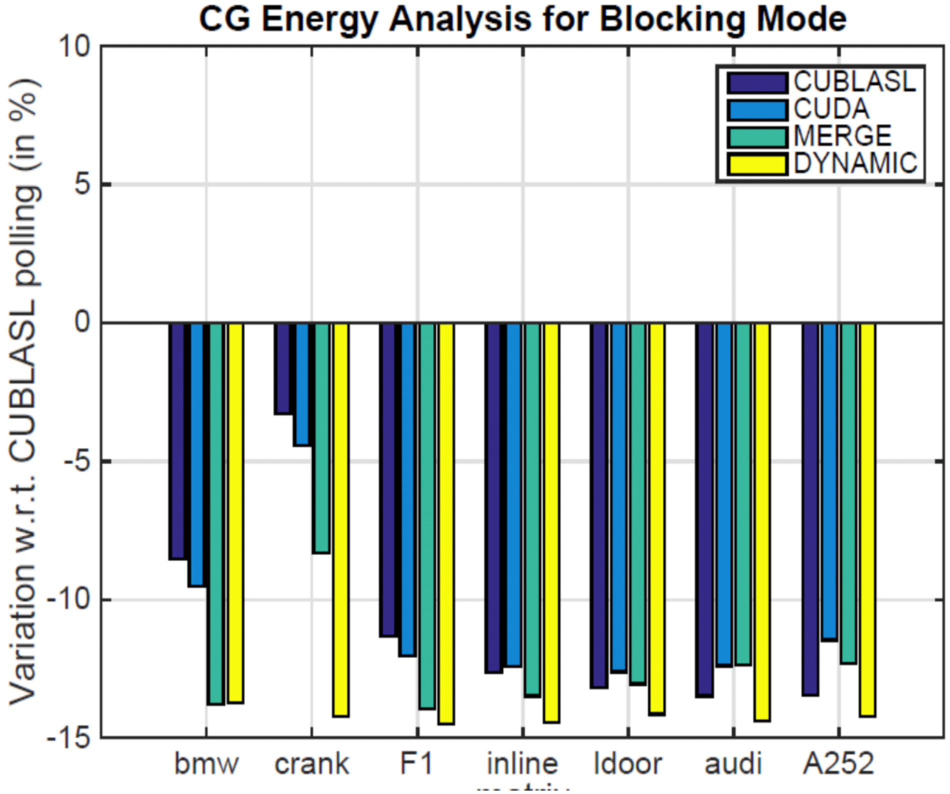




Energy saving for GPU servers
Avoiding polling CPU

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





Energy saving for GPU servers
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?

Exa2Green
energy-aware numerics

EU Ref. #318793

PA-HPC
Power-Aware High Performance Computing

TIN2011-23283