UNIVERSITAT JAUME·I

Energy-Aware Dense and Sparse Linear Algebra

Enrique S. Quintana-Ortí

UNIVERSITAT JAUME·I HPC&A High Performance Computing and Architectures

Energy-Aware Dense and Sparse Linear Algebra (or… Doing Nothing to Save Energy in Matrix Computations)

Enrique S. Quintana-Ortí

HPC&A
High Performance Computing and Architectures

Concurrency and energy efficiency

20.1 PFLOPS (10^{15} flops/sec.)

2012 DOE/NNSA/LLNL Sequoia

- 10^9 core level
  (Power BQC, 1.60 GHz → 12.8 GFLOPS)
- 10^1 node level
  (16 cores/node)
- 10^5 cluster level
  (98.304 nodes)

Concurrency and energy efficiency

UNIVERSITAT JAUME·I

20.1 PFLOPS (10^{15} flops/sec.)

2012 DOE/NNSA/LLNL Sequoia

- 10^9 core level
  (Power BQC, 1.60 GHz → 12.8 GFLOPS)
- 10^1 node level
  (16 cores/node)
- 10^5 cluster level
  (98.304 nodes)

2020 EFLOPS (10^{18} flops/sec.)

- $10^{9.5}$ core level
- 10^3 node level!
- $10^{5.5}$ cluster level

Concurrency and energy efficiency

- Green500 vs Top500 (November 2012)

Rank Green/Top	Site	#Cores	MFLOPS/W
1/253	National Institute for Computational Sciences/University of Tennessee	9.216 (Intel Xeon E5) 8.640 (Intel Xeon Phi)	2,499.44
3/1	DOE/SC/Oak Ridge National Laboratory	560,640 (AMD Opteron) 261,632 (NVIDIA K20)	2,142.77

Concurrency and energy efficiency

- Green500 vs Top500 (November 2012)

Rank Green/Top	Site	#Cores	MFLOPS/W	MW to EXAFLOPS?
1/253	National Institute for Computational Sciences/University of Tennessee	9.216 (Intel Xeon E5) 8.640 (Intel Xeon Phi)	2,499.44	400.09
3/1	DOE/SC/Oak Ridge National Laboratory	560,640 (AMD Opteron) 261,632 (NVIDIA K20)	2,142.77	466.69

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)

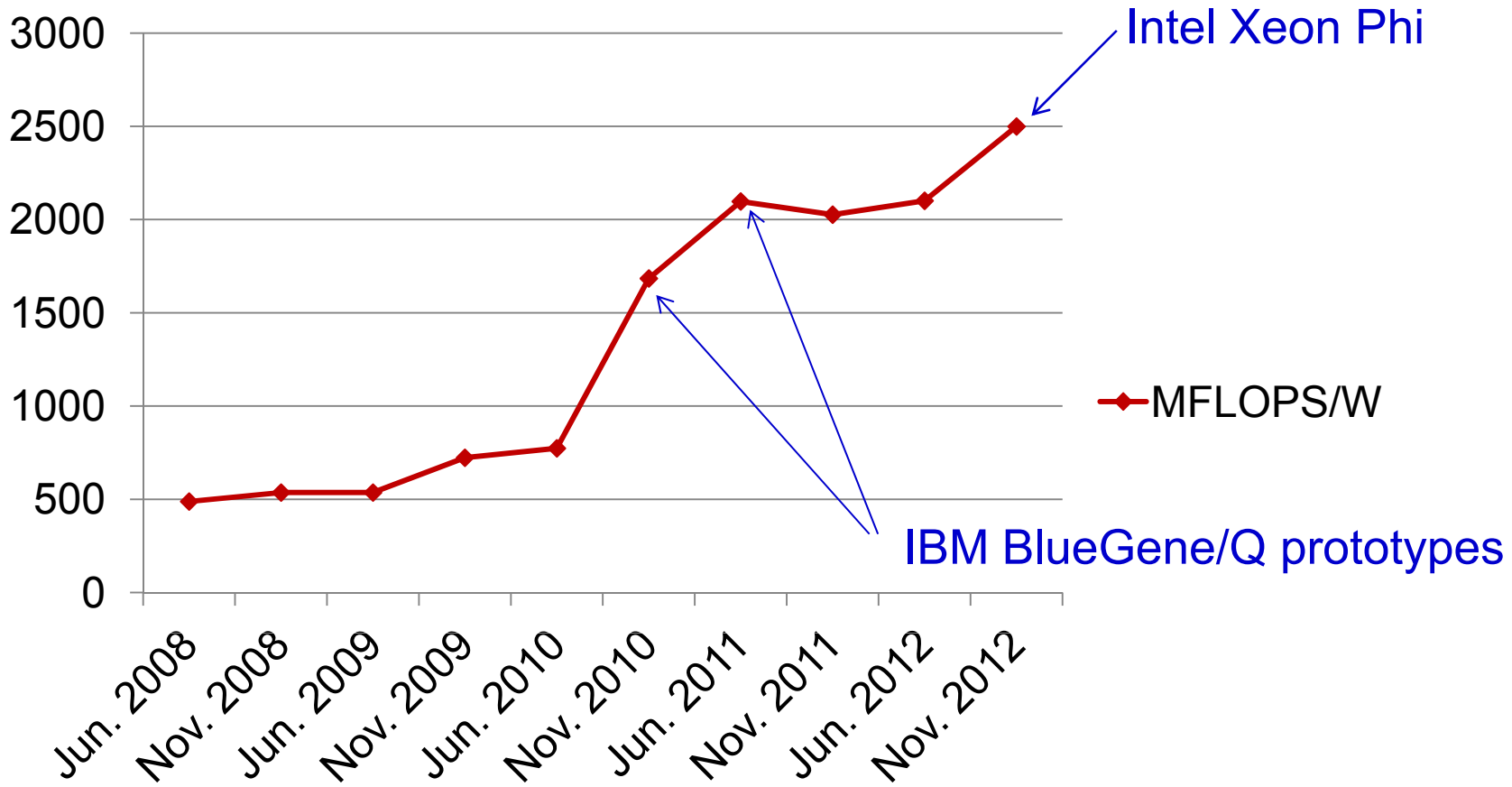
Rank Green/Top	Site	#Cores	MFLOPS/W	MW to EXAFLOPS?
1/253	National Comput Science Tenness		2,499.44	400.09
3/1	DOE/SC/Oak Ridge National Laboratory	560,640 (AMD Opteron) 261,632 (NVIDIA K20)	2,142.77	466.69

400 MW ≈ 315 MEuro/year!
(visit xe.com for £ ;-)

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

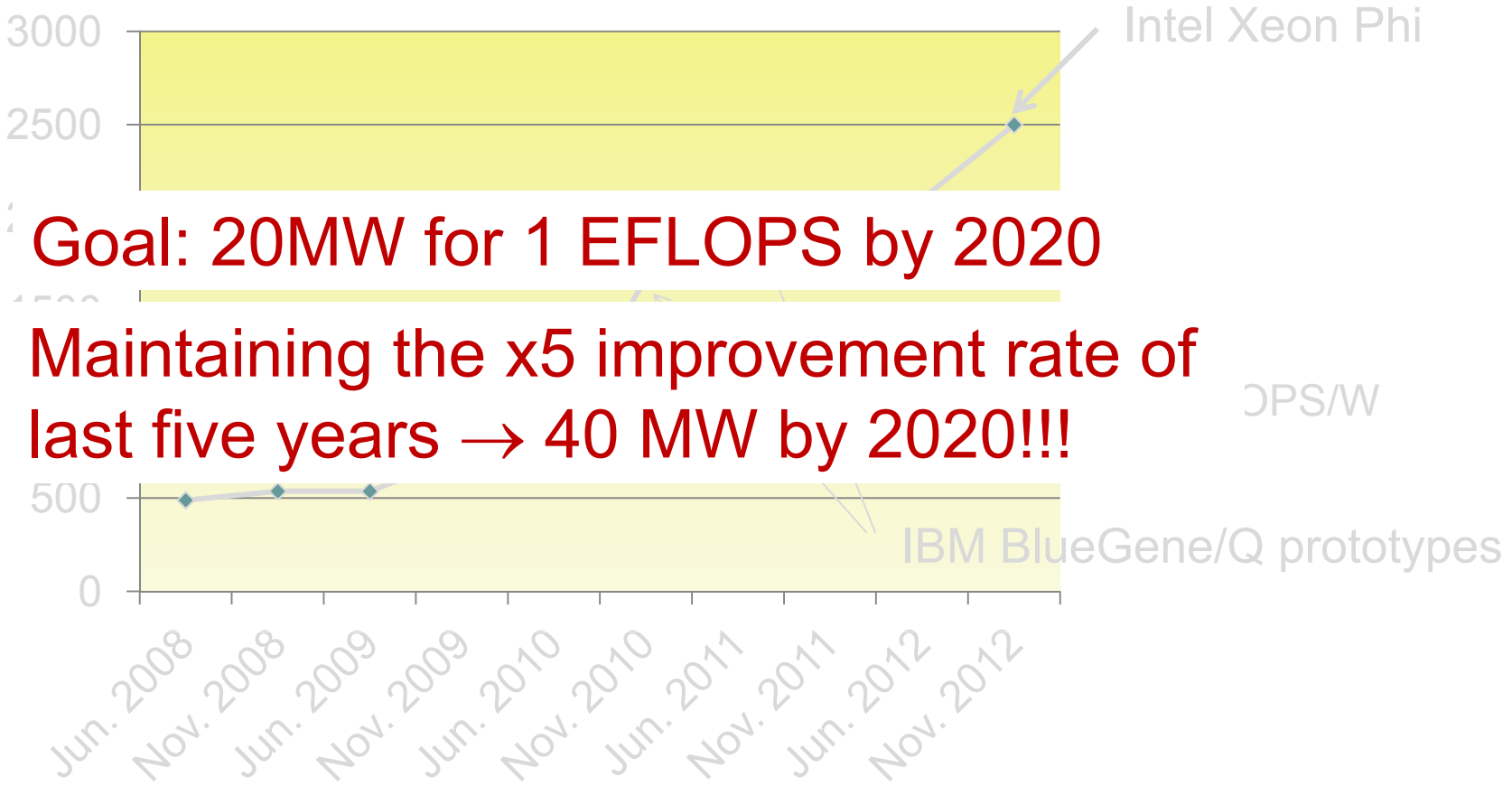
Concurrency and energy efficiency

- Systems ranked #1 in Green500



Concurrency and energy efficiency

- Systems ranked #1 in Green500



Goal: 20MW for 1 EFLOPS by 2020

Maintaining the x5 improvement rate of last five years → 40 MW by 2020!!!

Concurrency and energy efficiency

UNIVERSITAT JAUME·I

- 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 hw. reliability
- Personal view
  - Hardware features energy saving mechanisms
  - Scientific apps. are in general energy-oblivious

Numerical analysis and energy efficiency

UNIVERSITAT JAUME·I

- Trading stability for performance
  - Old idea, revisited many times
- Mixed precision
  - Also an old idea (Von Neumann), but interesting
- Probabilistic (or unreliable) systems-on-a-chip
- Energy-aware numerical algorithms

…

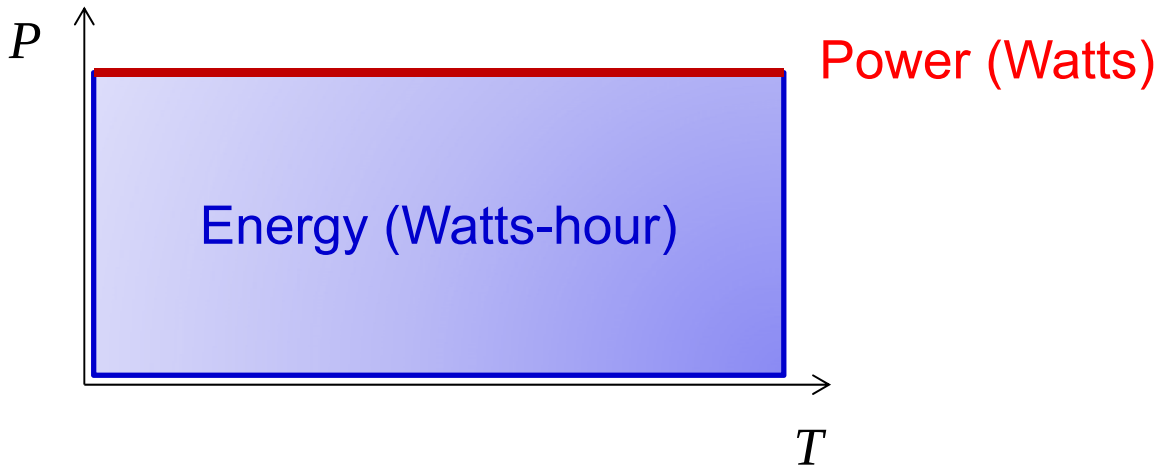
Outline

- Energy-aware numerical algorithms
  - ILUPACK for multicore processors
  - The CG method for hybrid CPU-GPU platforms

Basics…

- Remember:

$$E = \int_{0,T} P \; dt = P_{avq} \cdot T$$



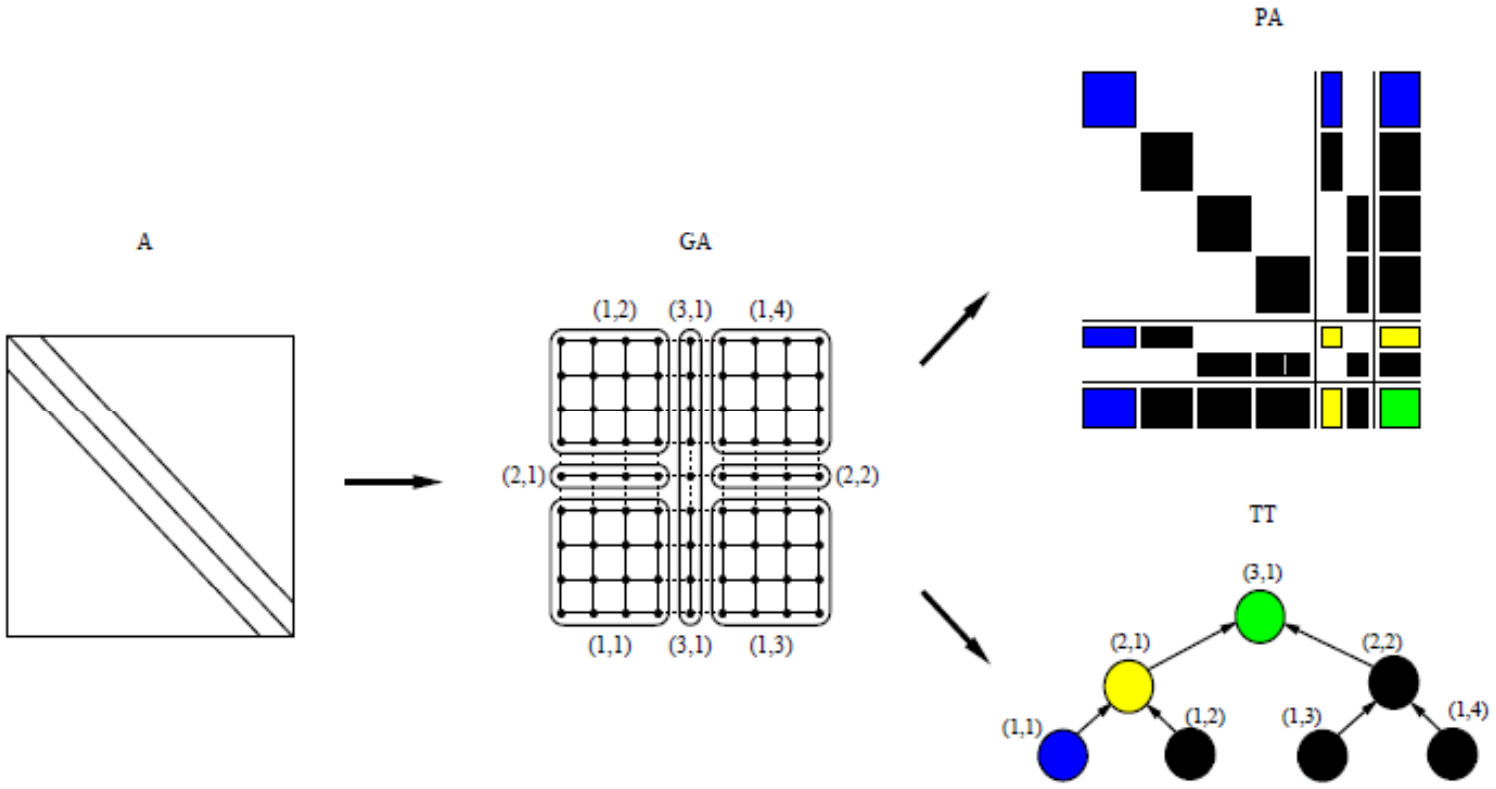
ILUPACK on multicore

UNIVERSITAT JAUME·I

- 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

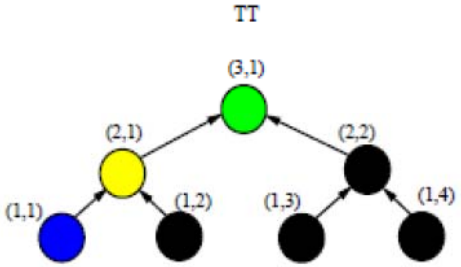
ILUPACK on multicore
Task parallelism

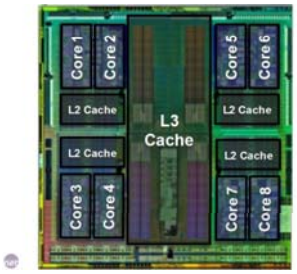
- Multi-threaded parallelism (real s.p.d. systems)
  - Leverage task parallelism



ILUPACK on multicore
Task parallelism

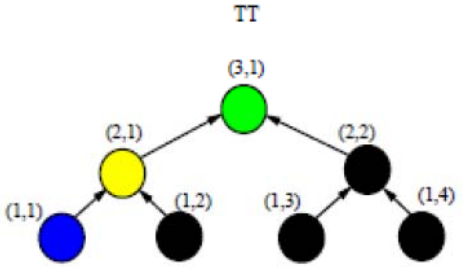
- Multi-threaded parallelism (real s.p.d. systems)
  - Out-of-order, dependency-aware execution of tasks

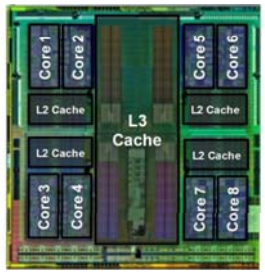




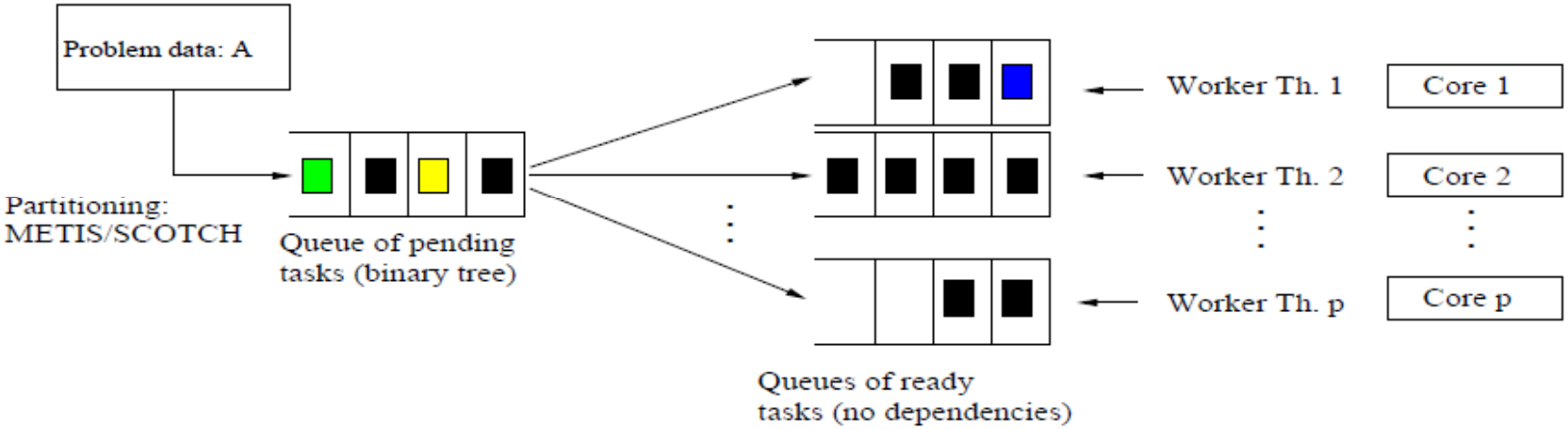
ILUPACK on multicore
Task parallelism

- Multi-threaded parallelism (real s.p.d. systems)
  - Out-of-order, dependency-aware execution of tasks



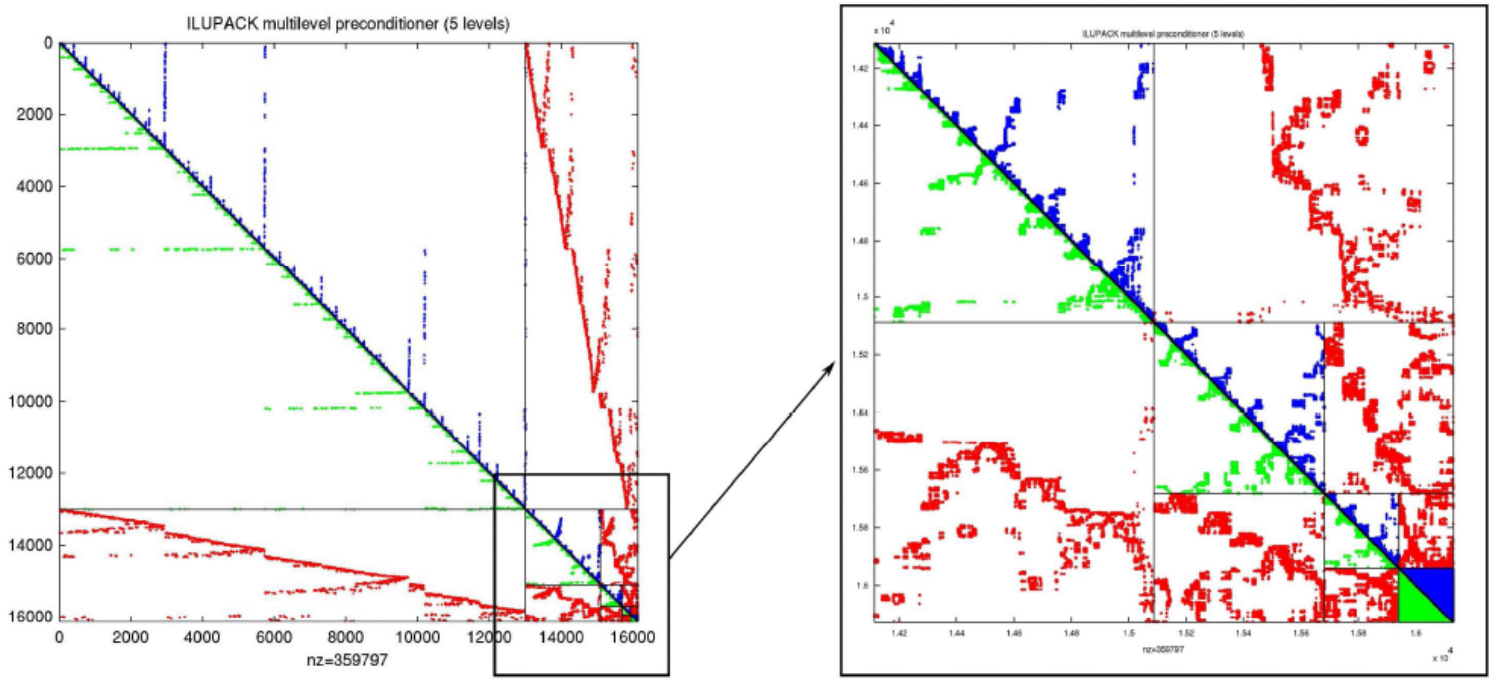


  - Dynamic scheduling via (OpenMP) runtime



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
Experimental setup

- CPU (performance) P-states:

AMD

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

P-state	V_{CC_i}	f_i
P_0	1.23	2.00
P_1	1.17	1.50
P_2	1.12	1.20
P_3	1.09	1.00
P_4	1.06	0.80

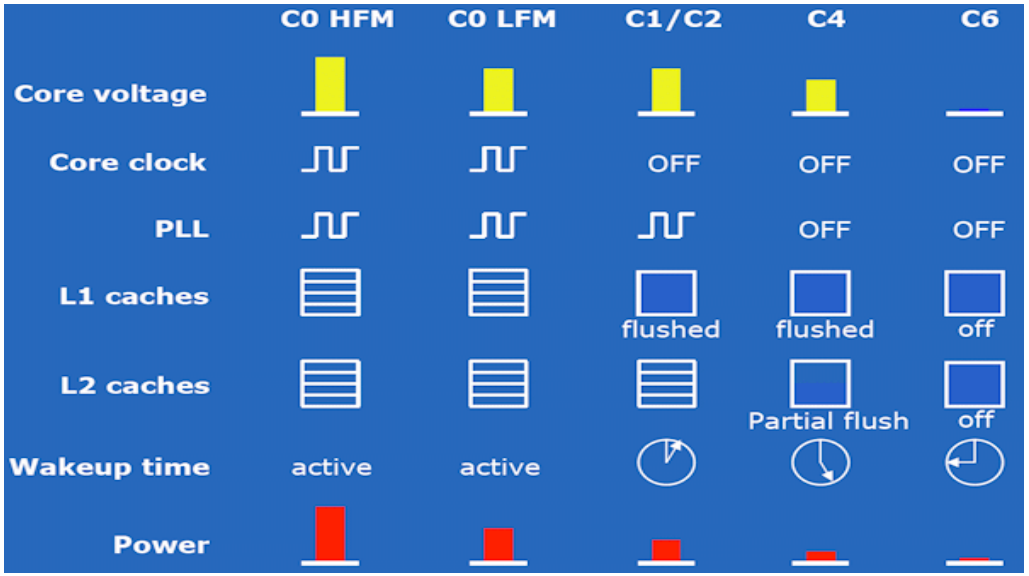
Intel

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

P-state	V_{CC_i}	f_i
P_0	1.04	2.00
P_1	0.98	1.73
P_2	0.95	1.60
P_3	1.01	1.87

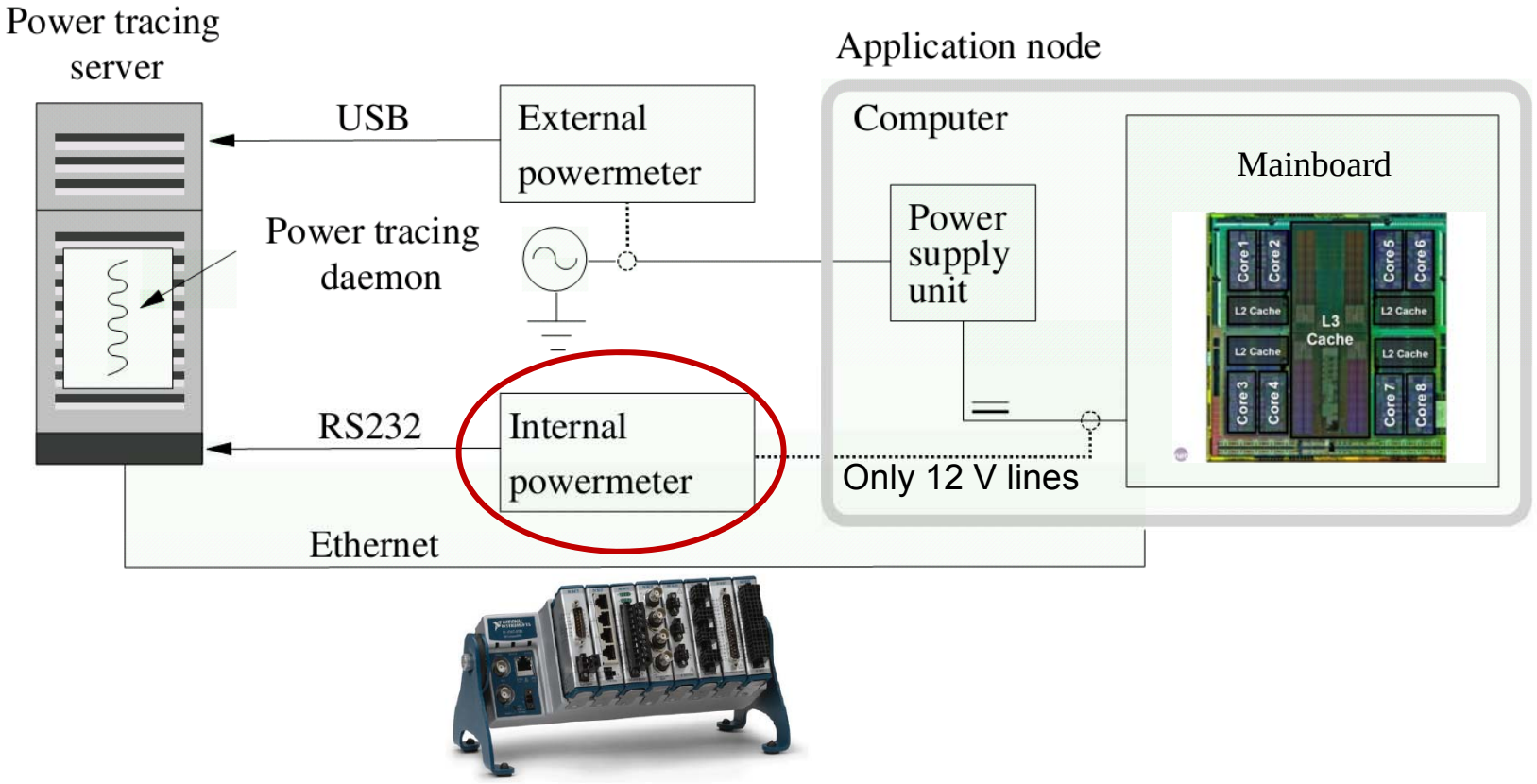
ILUPACK on multicore
Experimental setup

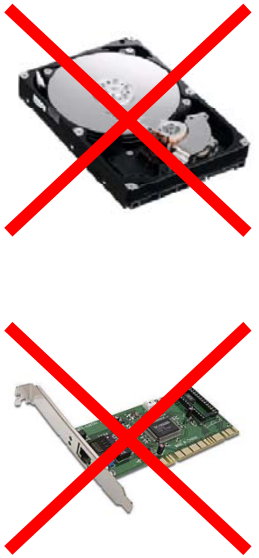
- CPU (power) C-states:
  - C0: normal operation mode
  - C1, C2,…: disable core components (L1/L2 caches), clock signal, memory controller,…
  - Trade off power for wakeup time



ILUPACK on multicore
Experimental setup

- National Instruments NI9205+NIcDAQ-9178
- 1,000 samples/sec. per channel





ILUPACK on multicore
Power model

$$P = P^{(S)Y(stem)} + P^{C(PU)} = P^{Y} + P^{S(tatic)} + P^{D(ynamic)}$$

- 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

ILUPACK on multicore
Power model (AMD)

- System power:

$$P = P^Y + P^S + P^D$$

Estimated as *idle* power

Due to off-chip components:

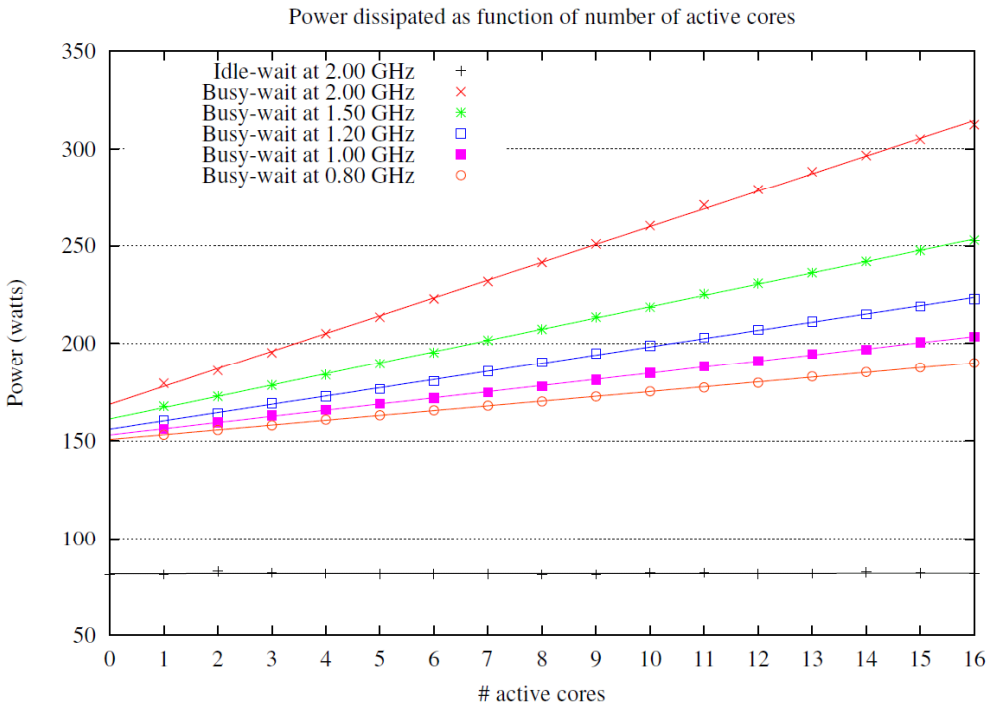
e.g., RAM (only mainboard)

$$P^Y \approx P^I = 80.15\,\text{W}$$

ILUPACK on multicore
Power model (AMD)

- Static power:

$$P = P^Y + P^S + P^D$$



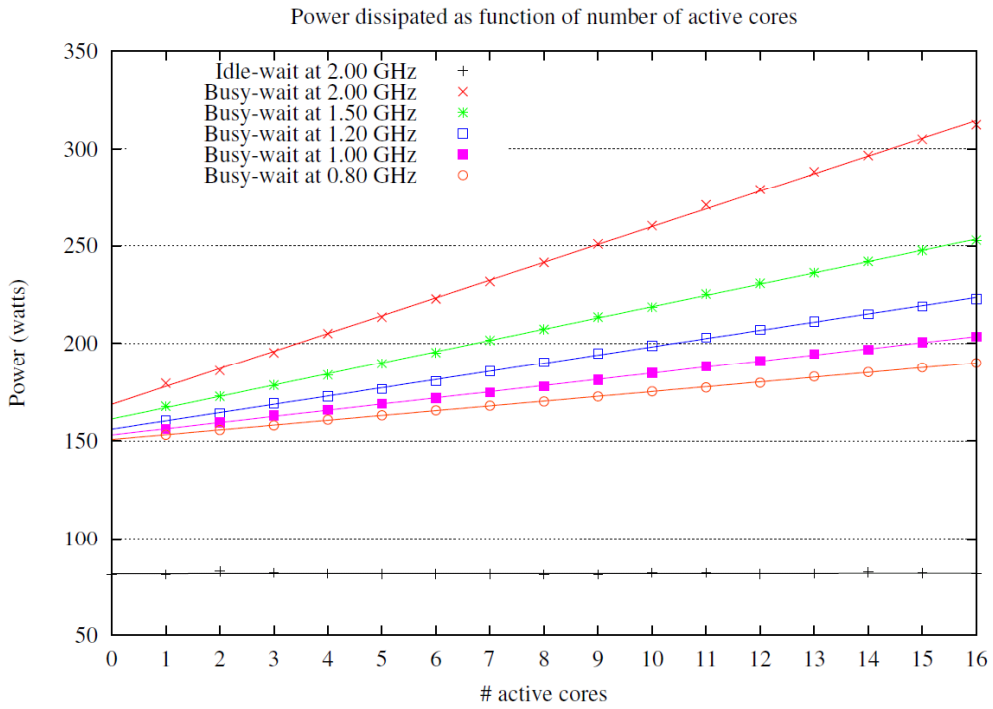
$$P^T{}_0(c) = a_0 + b_0 \cdot c = 168.59 + 9.12 \cdot c \text{ W}$$

$$P^S{}_0 \approx a_0 - P^Y = 88.44 \text{ W}$$

ILUPACK on multicore
Power model (AMD)

- Dynamic power:

$$P = P^Y + P^S + P^D$$



$$P^T{}_0(c) = a_0 + b_0 c = 168.59 + 9.12 \cdot c \text{ W}$$

Busy-wait: $P^D{}_0 \approx b_0 c = 9.12 \cdot c$ W

ILUPACK on multicore
Power model (AMD)

P-state P_i	Vcc_i	f_i	α_i	β_i	ΔP_i^S	ΔP_i^D
P_0	1.23	2.00	168.59	9.12	–	–
P_1	1.17	1.50	161.10	5.77	-9.52	-32.14
P_2	1.12	1.20	155.90	4.23	-17.09	-50.25
P_3	1.09	1.00	152.94	3.15	-21.47	-60.73
P_4	1.06	0.80	150.61	2.44	-25.73	-70.30

- P_i^S depends on Vcc_i^2

  E.g., P_0 (1.23V) $\rightarrow$ P_3 (1.09V): -21.47%

- P_i^D depends on $Vcc_i^2 \cdot f_i$

  E.g., P_0 (1.23V, 2.00GHz) $\rightarrow$ P_3 (1.09V, 1.00GHz): -60.73%

- These values agree within 2.5% with the experimental linear regression models!

ILUPACK on multicore
Leveraging P-states (AMD)

P-state P_i	Vcc_i	f_i	α_i	β_i	ΔP_i^S	ΔP_i^D
P_0	1.23	2.00	168.59	9.12	–	–
P_1	1.17	1.50	161.10	5.77	-9.52	-32.14
P_2	1.12	1.20	155.90	4.23	-17.09	-50.25
P_3	1.09	1.00	152.94	3.15	-21.47	-60.73
P_4	1.06	0.80	150.61	2.44	-25.73	-70.30

- 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

P-state P_i	T_i	$\bar{P}_i^T$	E_i	ΔT_i	$\Delta \bar{P}_i^T$	ΔE_i
P_0	34.06	282.87	9,634.78	–	–	–
P_1	43.57	235.64	10,267.72	21.88	-16.69	6.53
P_2	54.48	210.86	11.478.79	59.91	-25.45	19.20
P_3	61.58	197.01	12.132.79	80.73	-30.35	25.87
P_4	76.50	186.86	14,295.18	124.47	-33.94	48.28

Why?
Is really ILUPACK a memory-bounded operation?

ILUPACK on multicore
Leveraging P-states (AMD)

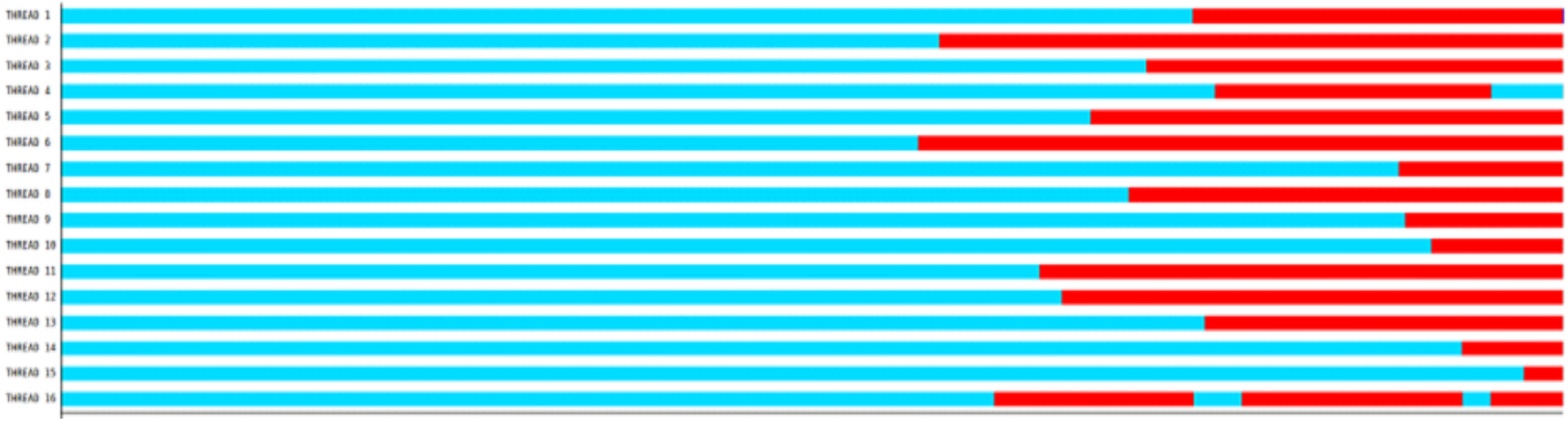
- 1st attempt: ~~Dynamic~~ Static voltage-frequency scaling

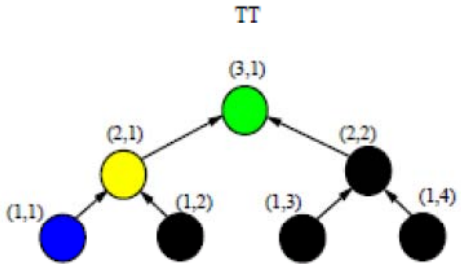
P-state P_i	Vcc_i	f_i	T_i	ΔT_i	BW_i	ΔBW_i
P_0	1.23	2.00	34.06	–	30.29	–
P_1	1.17	1.50	43.57	21.88	24.63	-18.67
P_2	1.12	1.20	54.48	59.91	20.46	-32.44
P_3	1.09	1.00	61.58	80.73	17.48	-42.30
P_4	1.06	0.80	76.50	124.47	14.00	-53.77

- Combined effect of linear decrease of CPU performance and memory bandwidth!
- Decrease of $P^S{}_i$ ($P_0 \rightarrow P_3$: -21.47%), decrease of $P^D{}_i$ ($P_0 \rightarrow P_3$: -60.73%) but P^Y does not change!

ILUPACK on multicore
Leveraging P-states (AMD)

- 2nd attempt: DVFS during idle periods

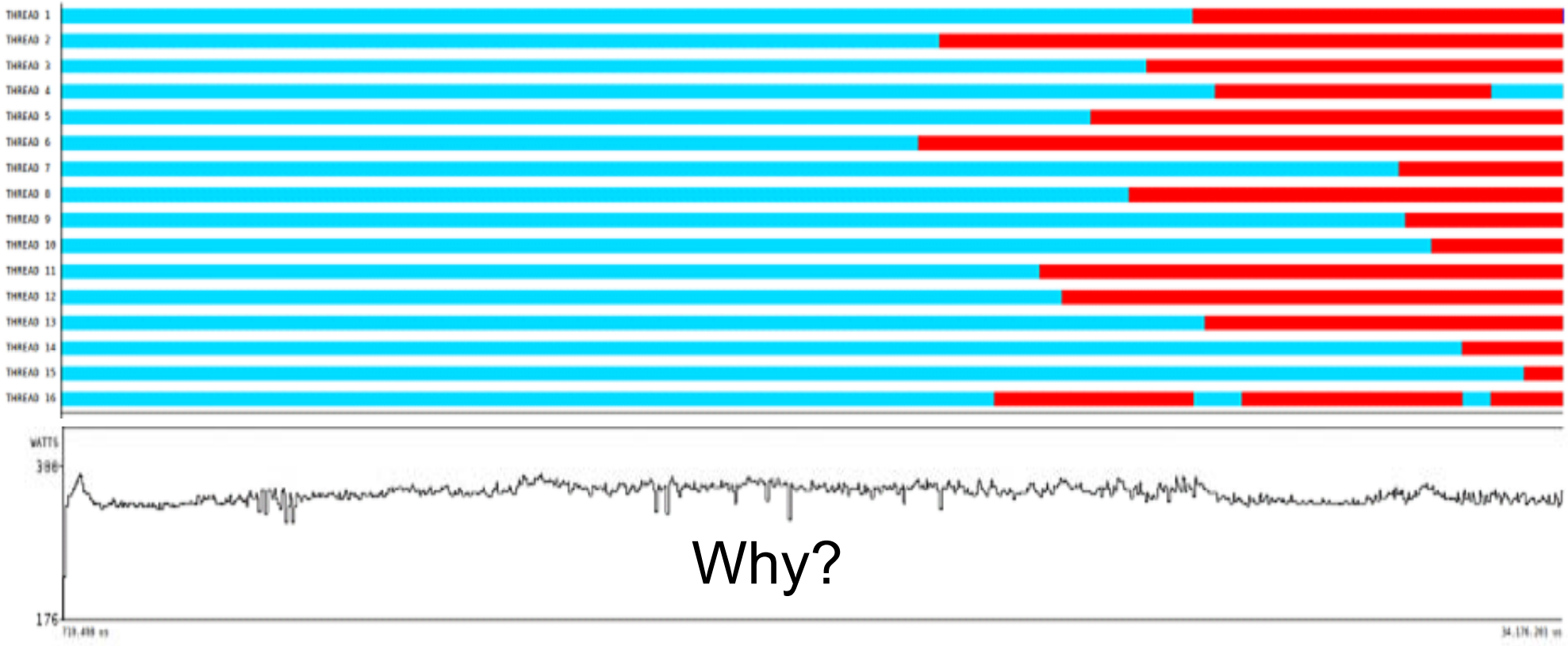




Calculation of preconditioner

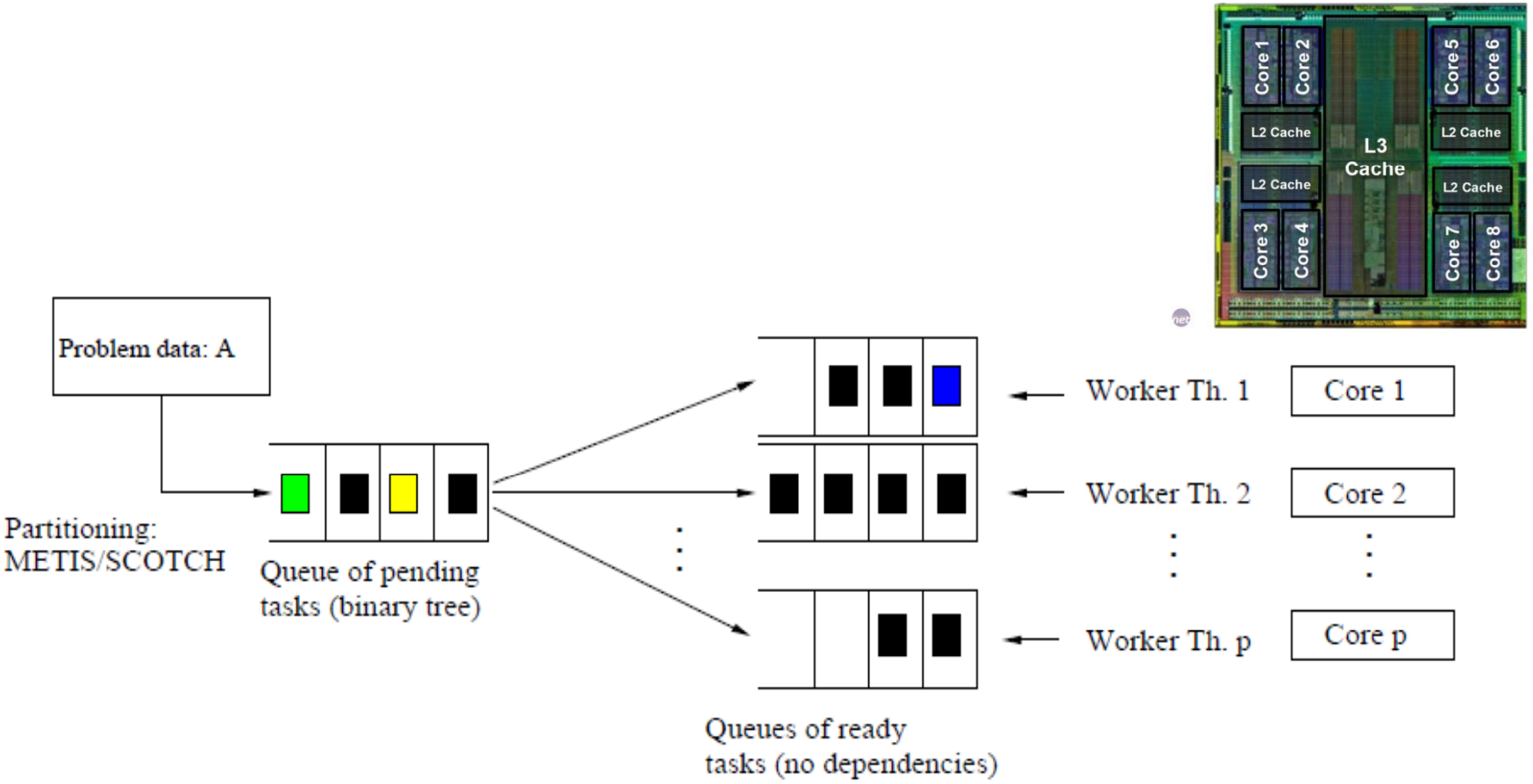
ILUPACK on multicore
Leveraging P-states (AMD)

- 2nd attempt: DVFS during idle periods



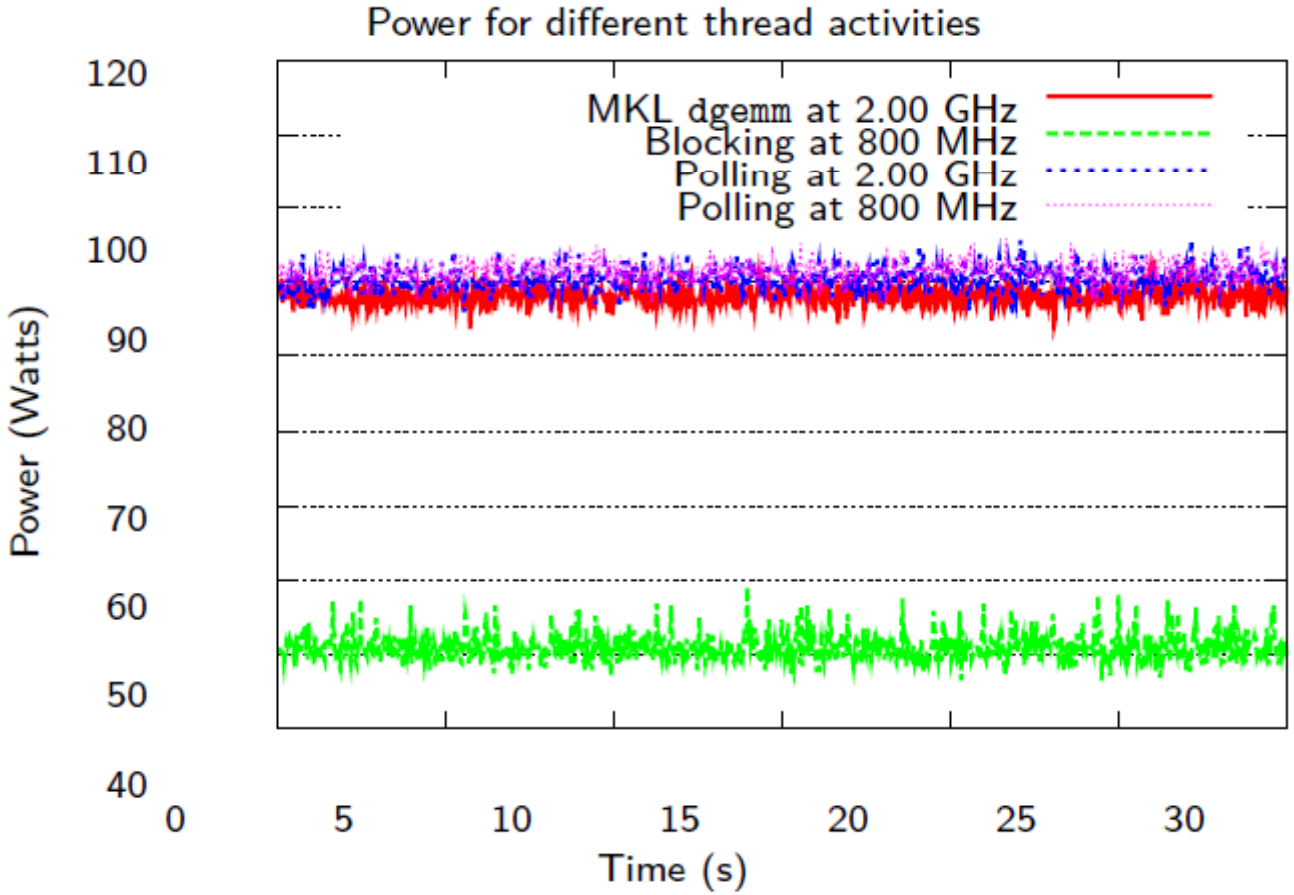
ILUPACK on multicore Leveraging P-states (AMD)

- 2nd attempt: DVFS during idle periods



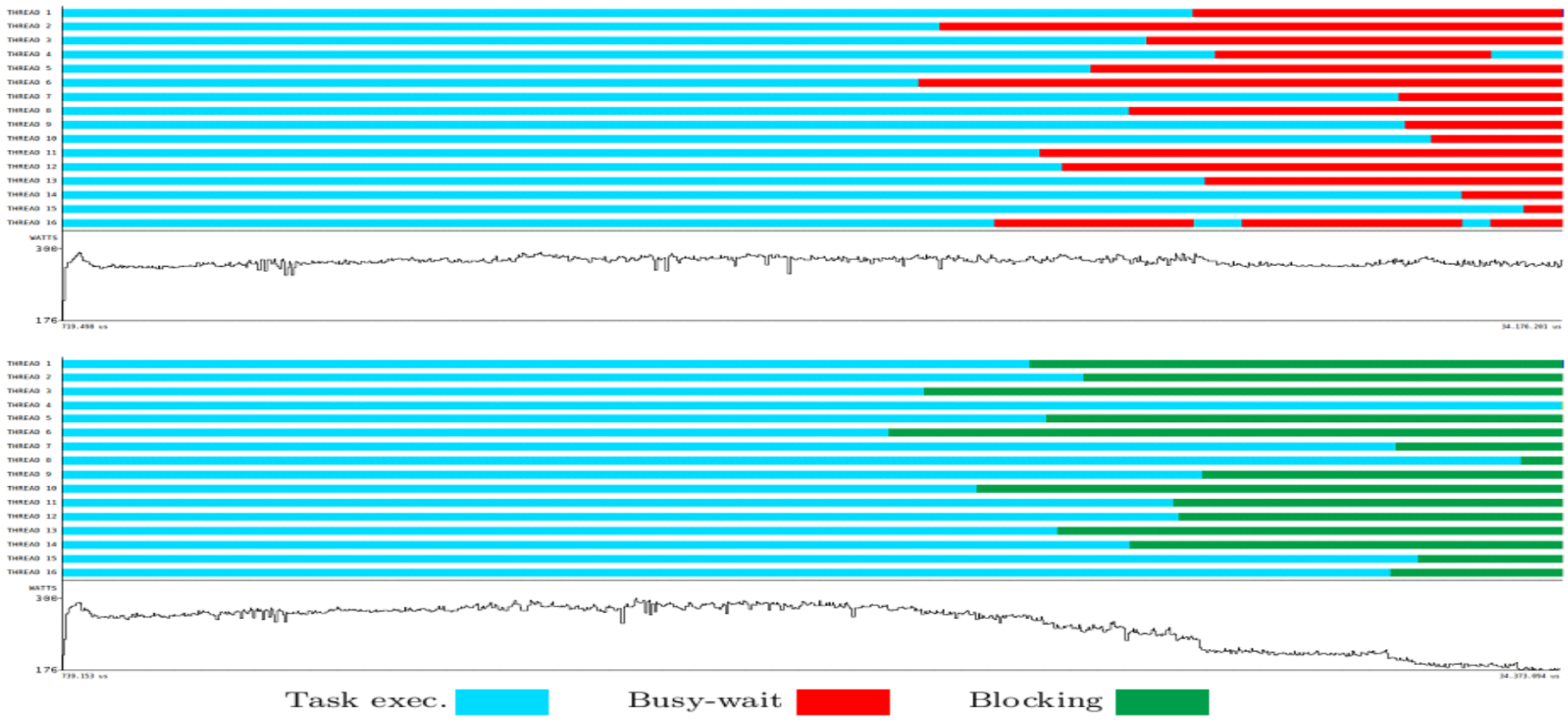
ILUPACK on multicore
Leveraging P-states (AMD)

- Active polling for work…



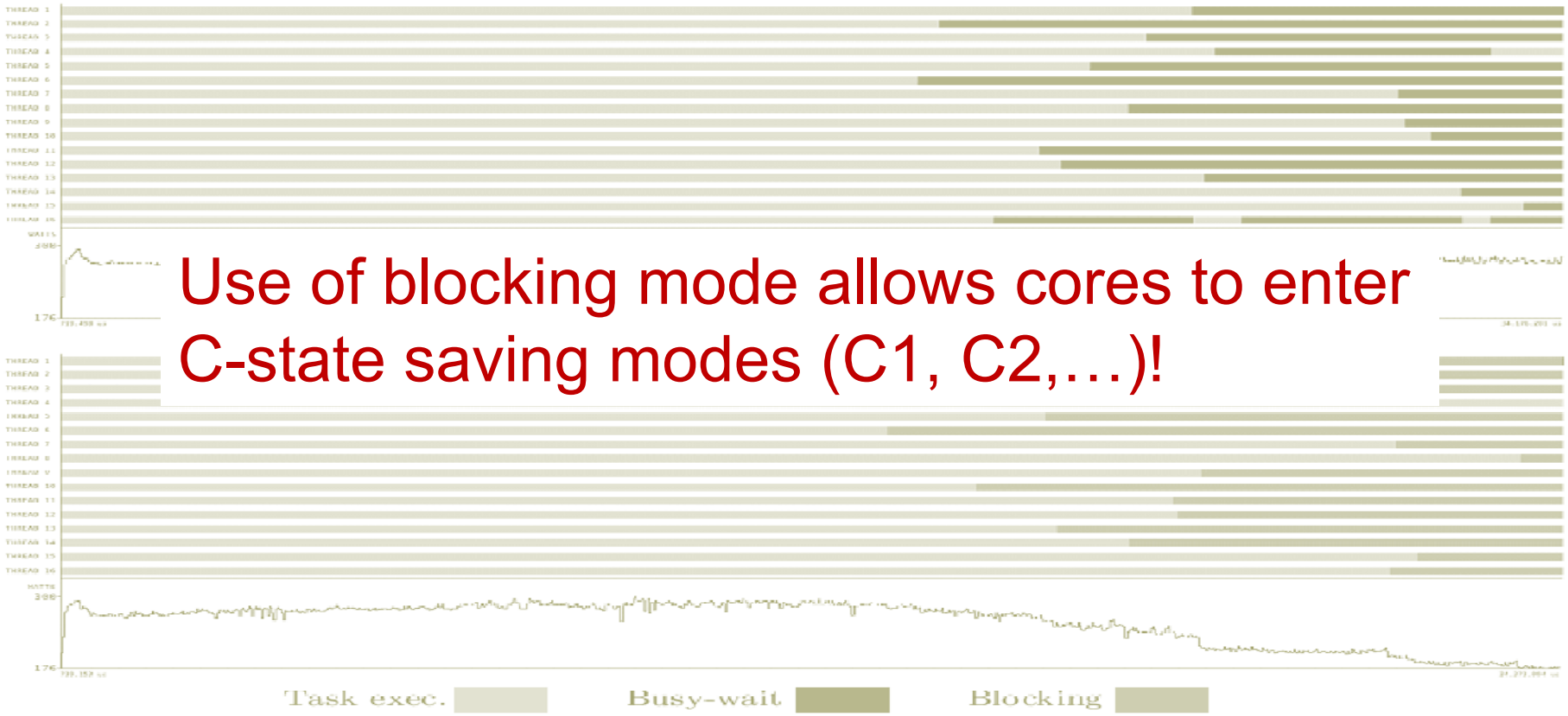
ILUPACK on multicore
Leveraging P- and C-states (AMD)

- 3rd attempt: DVFS and idle-wait



ILUPACK on multicore
Leveraging P- and C-states (AMD)

- 3rd attempt: DVFS and idle-wait



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: 39.32%
  - Upper bound of savings: $39.32 \cdot 0.2370 \approx 9.32\%$

ILUPACK on multicore
Leveraging P-states (Intel)

P-state	VCC_i	f_i	BW_i	ΔBW_i
P_0	1.04	2.00	12.72	–
P_1	0.98	1.73	12.61	-0.87
P_2	0.95	1.60	12.55	-1.34
P_3	1.01	1.87	12.58	-1.10

DVFS

	P_i	T_i	$\bar{P}_i^T$	E_i	ΔT_i	$\Delta \bar{P}_i^T$	ΔE_i
ILU	P_0	56.43	135.17	7,627.97	–	–	–
	P_1	59.06	127.96	7,557.87	4.67	-5.33	-0.92
	P_2	62.93	121.99	7,676.98	11.52	-9.75	0.64
	P_3	67.05	116.22	7,792.77	18.82	-18.82	2.16
Solve	P_0	148.94	155.27	23,123.99	–	–	–
	P_1	148.52	151.07	22,434.73	-0.28	-2.70	-2.98
	P_2	154.86	145.11	22,469.38	3.97	-6.55	-2.83
	P_3	159.08	138.50	22,033.14	6.81	-10.80	-4.72

DVFS per socket, not per core!

ILUPACK on multicore
Leveraging P- and C-states (Intel)

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

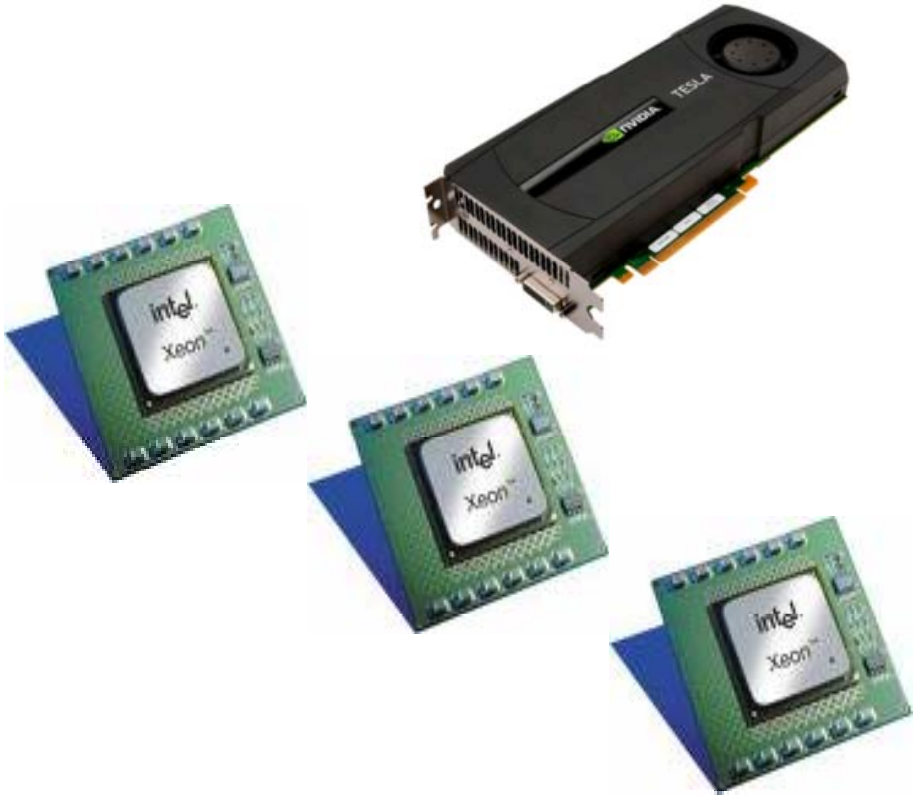
DVFS

	P_i	T_i	$\bar{P}_i^T$	E_i	ΔT_i	$\Delta \bar{P}_i^T$	ΔE_i
ILU	P_0	56.43	135.17	7,627.97	–	–	–
	P_1	59.06	127.96	7,557.87	4.67	-5.33	-0.92
	P_2	62.93	121.99	7,676.98	11.52	-9.75	0.64
	P_3	67.05	116.22	7,792.77	18.82	-18.82	2.16
Solve	P_0	148.94	155.27	23,123.99	–	–	–
	P_1	148.52	151.07	22,434.73	-0.28	-2.70	-2.98
	P_2	154.86	145.11	22,469.38	3.97	-6.55	-2.83
	P_3	159.08	138.50	22,033.14	6.81	-10.80	-4.72

DVFS+idle-wait

	P_i	T_i	$\bar{P}_i^T$	E_i	ΔT_i	$\Delta \bar{P}_i^T$	ΔE_i
ILU	P_0	55.96	122.83	6,872.82	–	–	–
	P_1	59.50	116.52	6,932.86	6.34	-5.14	0.87
	P_2	62.37	112.42	7,012.13	11.46	-8.47	2.03
	P_3	66.84	107.34	7,174.81	19.46	-12.61	4.39
Solve	P_0	147.40	146.63	21,594.05	–	–	–
	P_1	148.41	143.18	21,241.34	0.68	-2.35	-1.63
	P_2	151.39	138.96	21,037.62	2.71	-5.23	-2.58
	P_3	158.28	132.65	20,992.39	7.38	-9.53	-2.79

The CG method on CPU-GPU



- GPU (or other hardware accelerator, e.g., Intel Xeon Phi):
  - High throughput
  - Reasonable power
  - Reasonable(?) cost
    - (3,000 Euro NVIDIA K20)
  - MFLOPS/W!

- Systems #1 in both green500 and top500!

The CG method on CPU-GPU

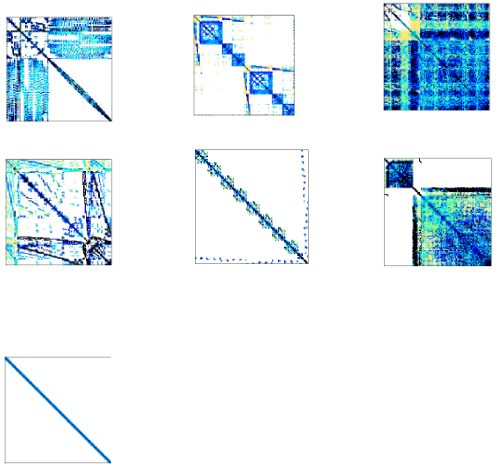
- Leveraging P-states on CPU-GPU platforms?
  - DVFS in the CPU while computation proceeds on the GPU?
- Leveraging C-states on CPU-GPU platforms?
  - What is the idle CPU doing?

The CG method on CPU-GPU
Experimental setup

- Platform:
  - Intel i7-3770K (Ivy Bridge), 16GB
  - NVIDIA GeForce GTX480 (Fermi)

- Cases from two matrix collections

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



The CG method on CPU-GPU
Basic implementation

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

```
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

```
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 reduce power of CPU!

The CG method on CPU-GPU
Basic implementation

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

```
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:

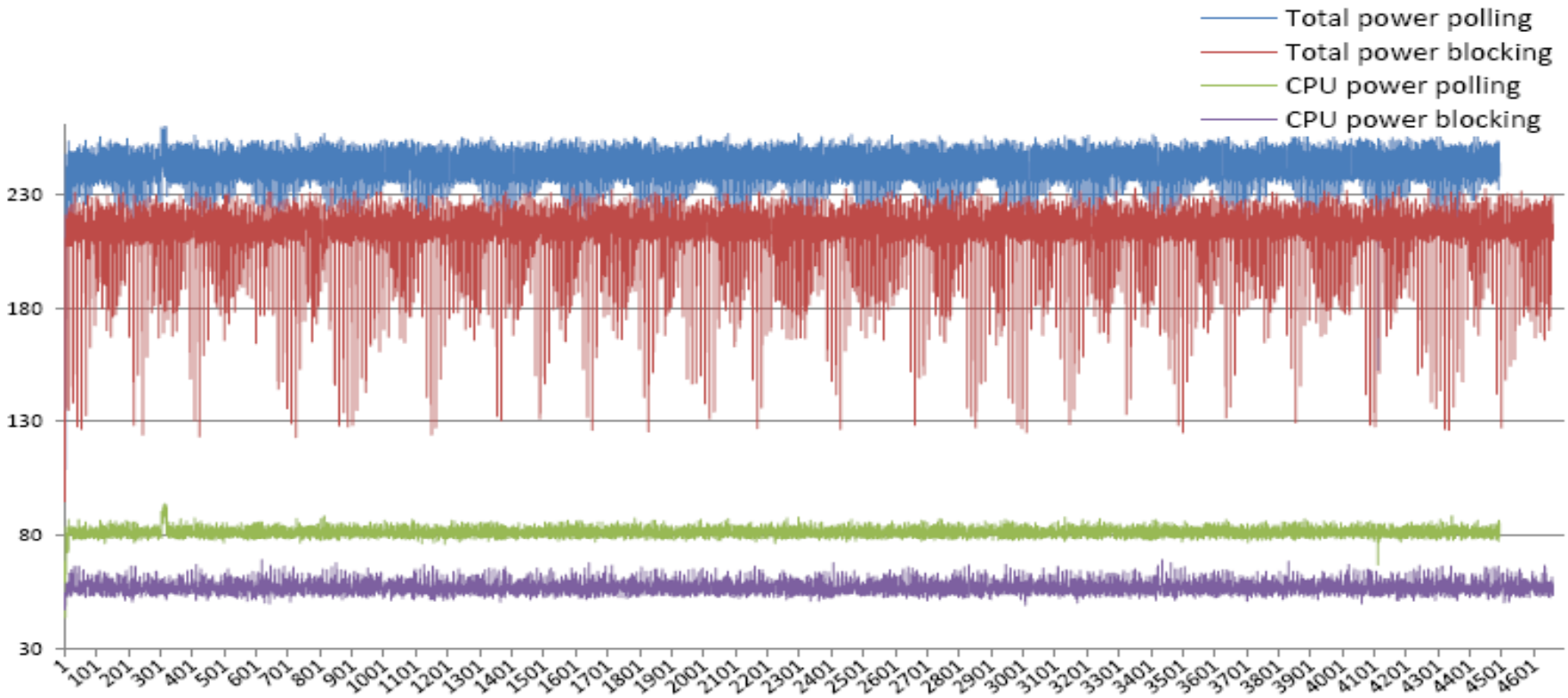
- What is the CPU doing while idle?
- CUDA offers polling (active-wait) vs blocking (idle-wait) operation modes

```
  cublasSaxpy (n, one, r, 1, d, 1 );
  res = sqrt( beta );
  k++;
} // end-while
```

The CG method on CPU-GPU
Basic implementation

- Trading off power for time:

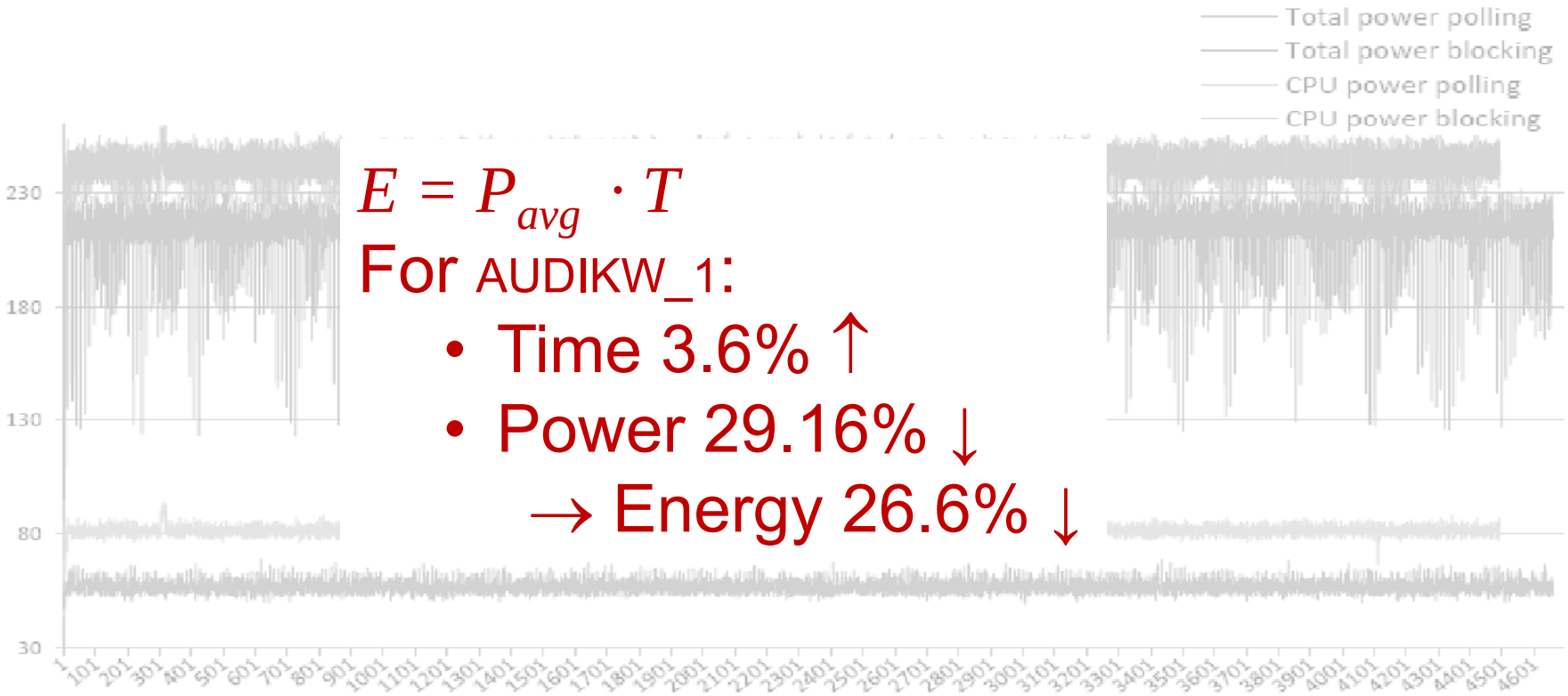
  CUDA blocking mode w.r.t. CUDA polling mode



The CG method on CPU-GPU
Basic implementation

- Trading off power for time:

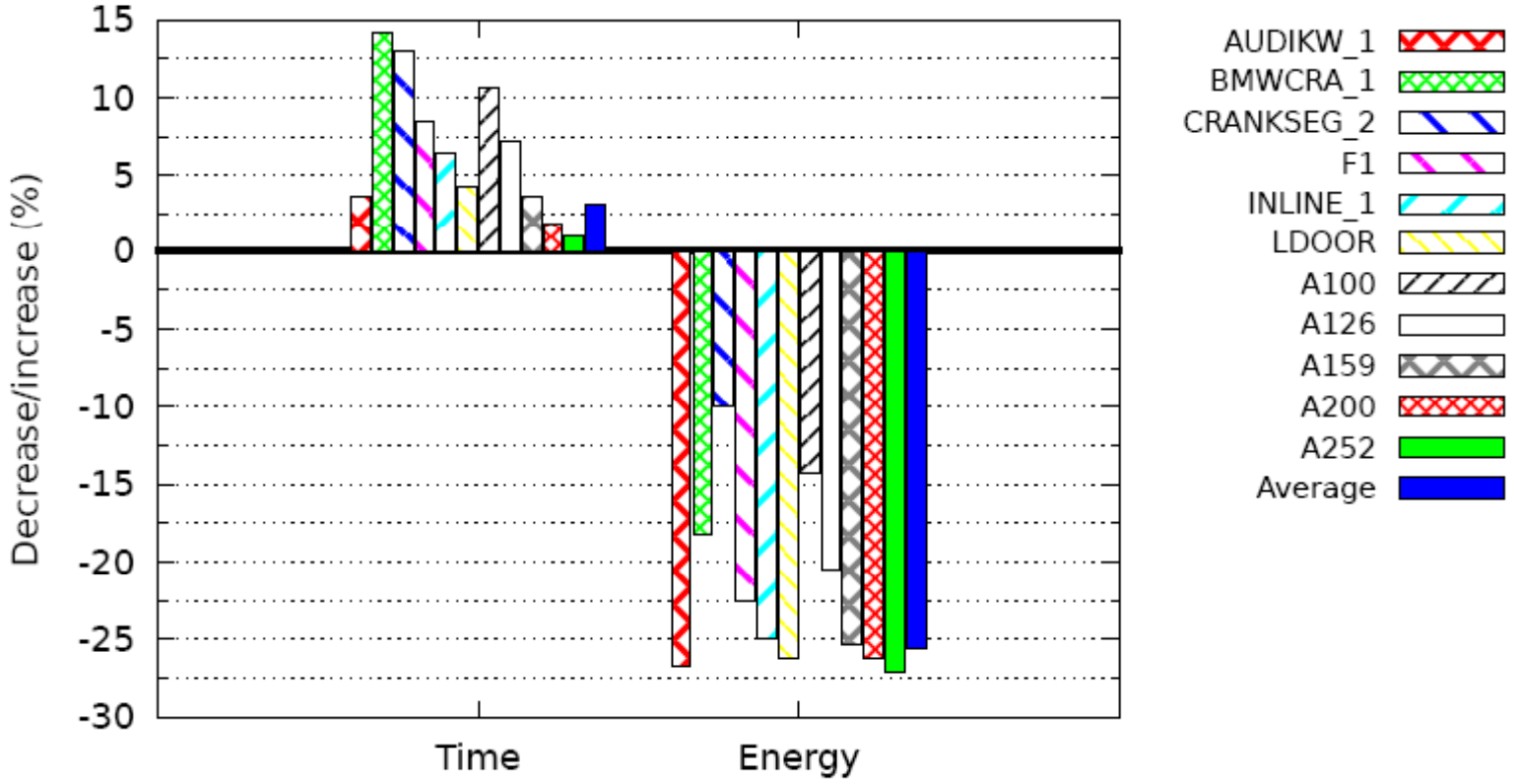
  CUDA blocking mode w.r.t. CUDA polling mode



The CG method on CPU-GPU
Basic implementation

- Trading off power for time:

  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)

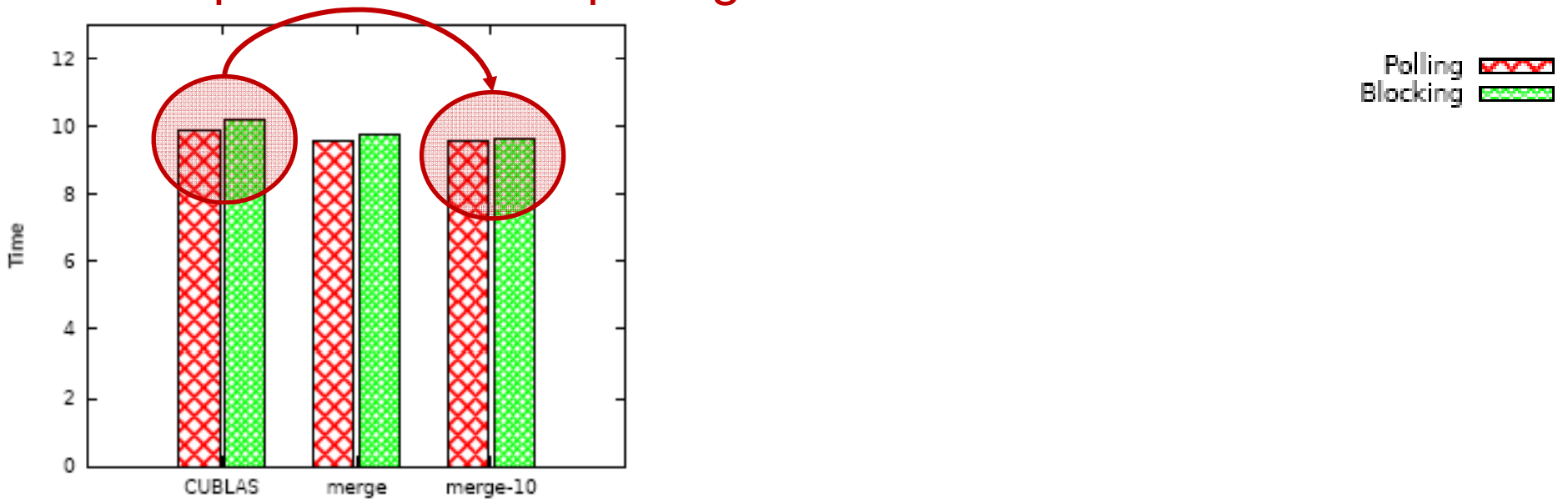
```
while( ( k < maxiter ) && ( res > epsilon ) ){
  SSpMV <<<Gs,Bs>>> ( n, rowA, colA, valA, d, z );
  tmp = cublasSdot ( n, d, 1, z, 1 );
  rho = beta / tmp;
  gamma = beta;
  cublasSaxpy (n,  rho, d, 1, x, 1 );
  cublasSaxpy (n, -rho, z, 1, r, 1 );
  tmp = cublasSdot ( n, r, 1, r, 1 );
  beta = tmp;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

```
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 );
  cudaMemcopy( &res, beta, sizeof(float),
                                  cudaMemcpyDeviceToHost );
  res = sqrt( beta );
  k ++;
} // end-while
```

The CG method on CPU-GPU
Merged implementation

- Time vs. CPU energy

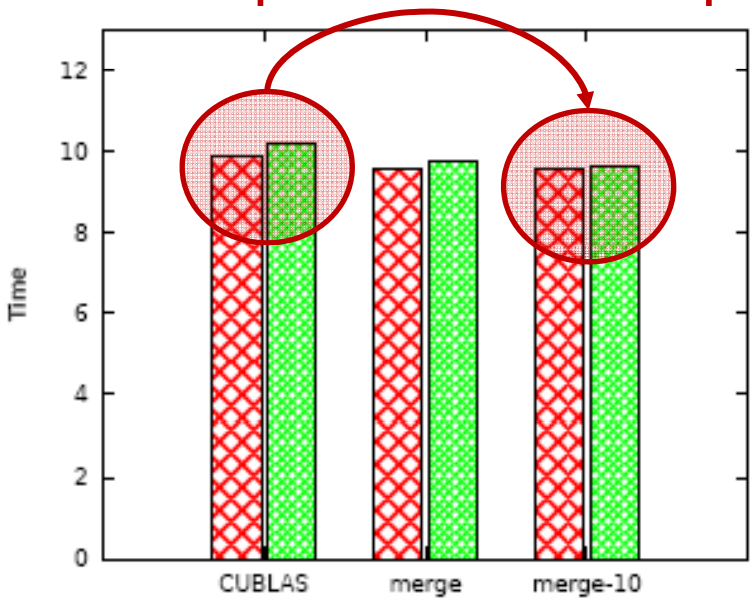
Maintain performance of polling…

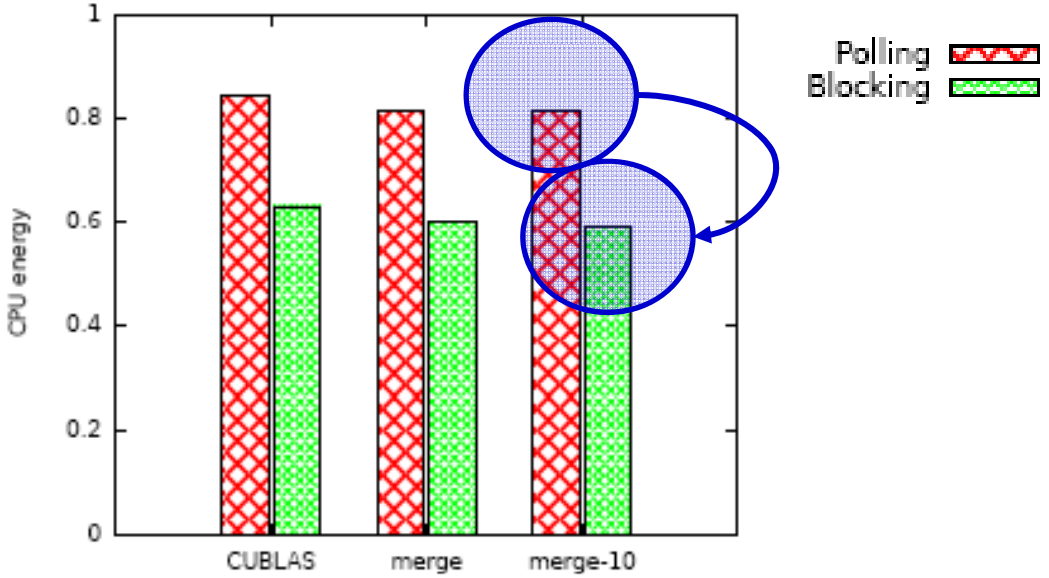


The CG method on CPU-GPU
Merged implementation

- Time vs. CPU energy

Maintain performance of polling…





…while leveraging power-efficiency
of C-states via CUDA blocking mode

Summary

- A battle to be fought in the core arena
  - More concurrency
  - Heterogeneous designs

Summary

- A related battle to be fought in the energy arena
  - P-states and C-states:

    "Doing nothing to save energy in matrix computations"

  - "*Do nothing, efficiently…*" (V. Pallipadi, A. Belay) or "*Doing nothing well*" (D. E. Culler)

  - Better (numerical) algorithms will reduce dynamic power but system power & static power are for the architects!

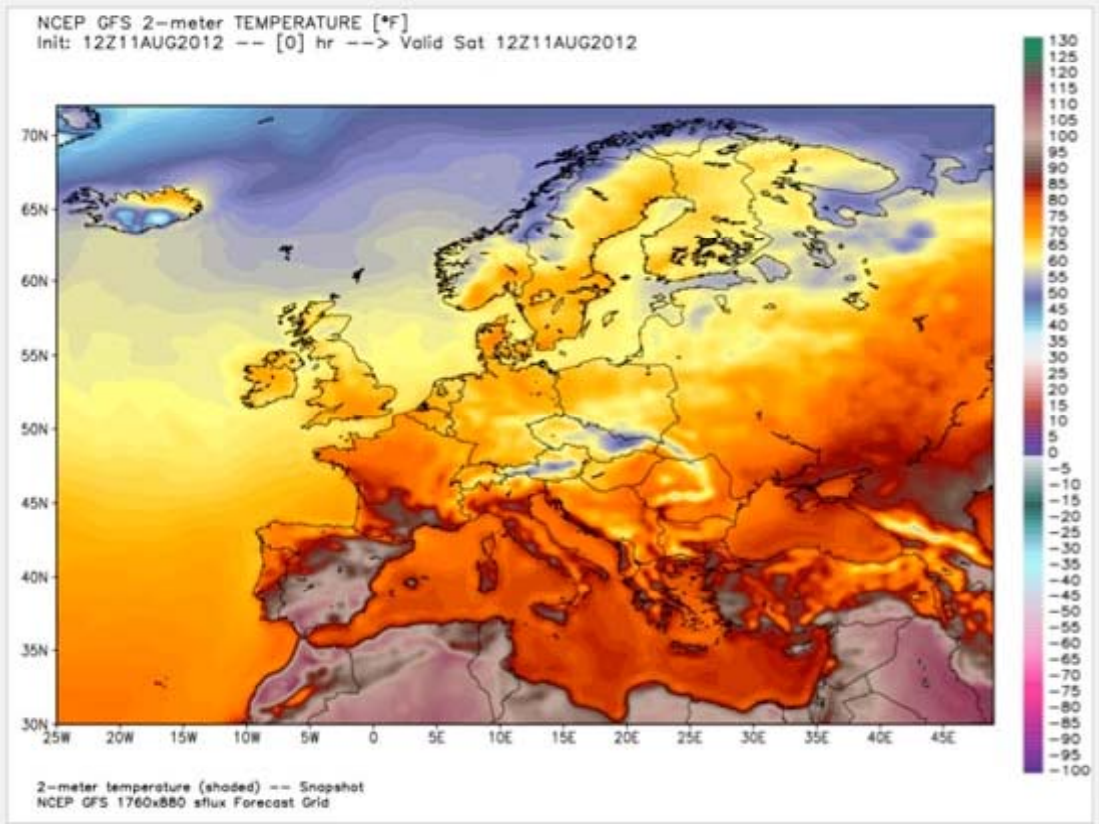
Summary

- Being energy-aware scientists?
  - Spring in Castellón: 21º C degrees (71º F) today!

Date	Fri 22			Sat 23				Sun 24		Mon 25		Mar 26	Wed 27	Thu 28
	6-12	12-18	18-24	0-6	6-12	12-18	18-24	0-12	12-24	0-12	12-24			
Sky conditions														
Prob Precip.	0%	0%	0%	15%	25%	15%	5%	25%	30%	5%	5%	25%	25%	15%
Snow bound prov. (M)					1800	1800		1700	1600				1700	1800
Temp. Min. / max. (° C)	10 / 21			9 / 19				9 / 18		11 / 21		12 / 21	14 / 21	14 / 21
Wind			–				–							–
(Km / h)	10	10	0	5	10	10	0	10	15	10	15	20	10	0
Maximum UV Index	6			4				4		5		5		
Notices	No Risk			No Risk										

Summary

- Being energy-aware scientists?
  - …but come back to Manchester for the summer



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"

More information

- "Leveraging task-parallelism in energy-efficient ILU preconditioners", J. I. Aliaga, M. F. Dolz, A. F. Martín, E. S. Quintana-Ortí. ICT-GLOW 2012. Vienna (Austria)

- "Modeling power and energy of the task-parallel Cholesky factorization on multicore processors", P. Alonso, M. F. Dolz, R. Mayo, E. S. Quintana-Ortí. EnaHPC 2012. Hamburg (Germany)

- "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

- Reformulated Conjugate Gradient for the Energy-Aware Solution of Linear Systems on GPUs. J. I. Aliaga, J. Pérez, E. S. Quintana-Ortí, ICPP 2013. Lyon (France). Submitted

More information

- "Leveraging task-parallelism in energy-efficient ILU preconditioners", J. I. Aliaga, M. F. Dolz, A. F. Martín, E. S. Quintana-Ortí. ICT-GLOW 2012. Vienna (Austria)

- "Modeling power and energy of the task-parallel Cholesky factorization on multicore processors", P. Alonso, M. F. Dolz, R. Mayo, E. S. Quintana-Ortí. EnaHPC 2012. Hamburg (Germany)

Thanks and... questions?

- "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

- Reformulated Conjugate Gradient for the Energy-Aware Solution of Linear Systems on GPUs. J. I. Aliaga, J. Pérez, E. S. Quintana-Ortí, ICPP 2013. Lyon (France). Submitted