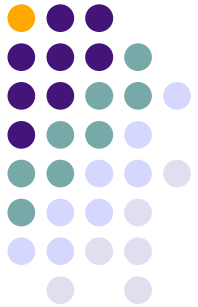
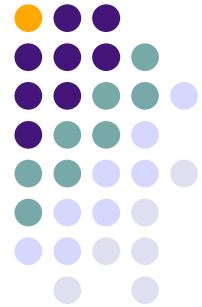


Performance and Energy Analysis of the Iterative Solution of Sparse Linear Systems on Multicore and Manycore Architectures

José I. Aliaga

Performance and Energy Analysis of the Iterative Solution of Sparse Linear Systems on Multicore and Manycore Architectures



- Universidad Jaime I (Castellón, Spain)
  - José I. Aliaga
  - Maribel Castillo
  - Juan C. Fernández
  - Germán León
  - Joaquín Pérez
  - Enrique S. Quintana-Ortí

UNIVERSITAT JAUME·I

ICL UT

THE UNIVERSITY of TENNESSEE

- Innovative and Computing Lab (Univ. Tennessee, USA)
  - Hartwig Antz

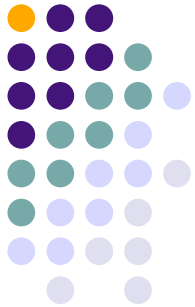
Concurrency and energy efficiency

2010 PFLOPS (10^{15} flops/sec.)

2010 JUGENE

- 10^9 core level
  (PowerPC 450, 850MHz → 3.4 GFLOPS)
- 10^1 node level
  (Quad-Core)
- 10^5 cluster level
  (73.728 nodes)

Concurrency and energy efficiency



2010 PFLOPS (10^{15} flops/sec.)

2010 JUGENE

- 10^9 core level
  (PowerPC 450, 850MHz → 3.4 GFLOPS)
- 10^1 node level
  (Quad-Core)
- 10^5 cluster level
  (73.728 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

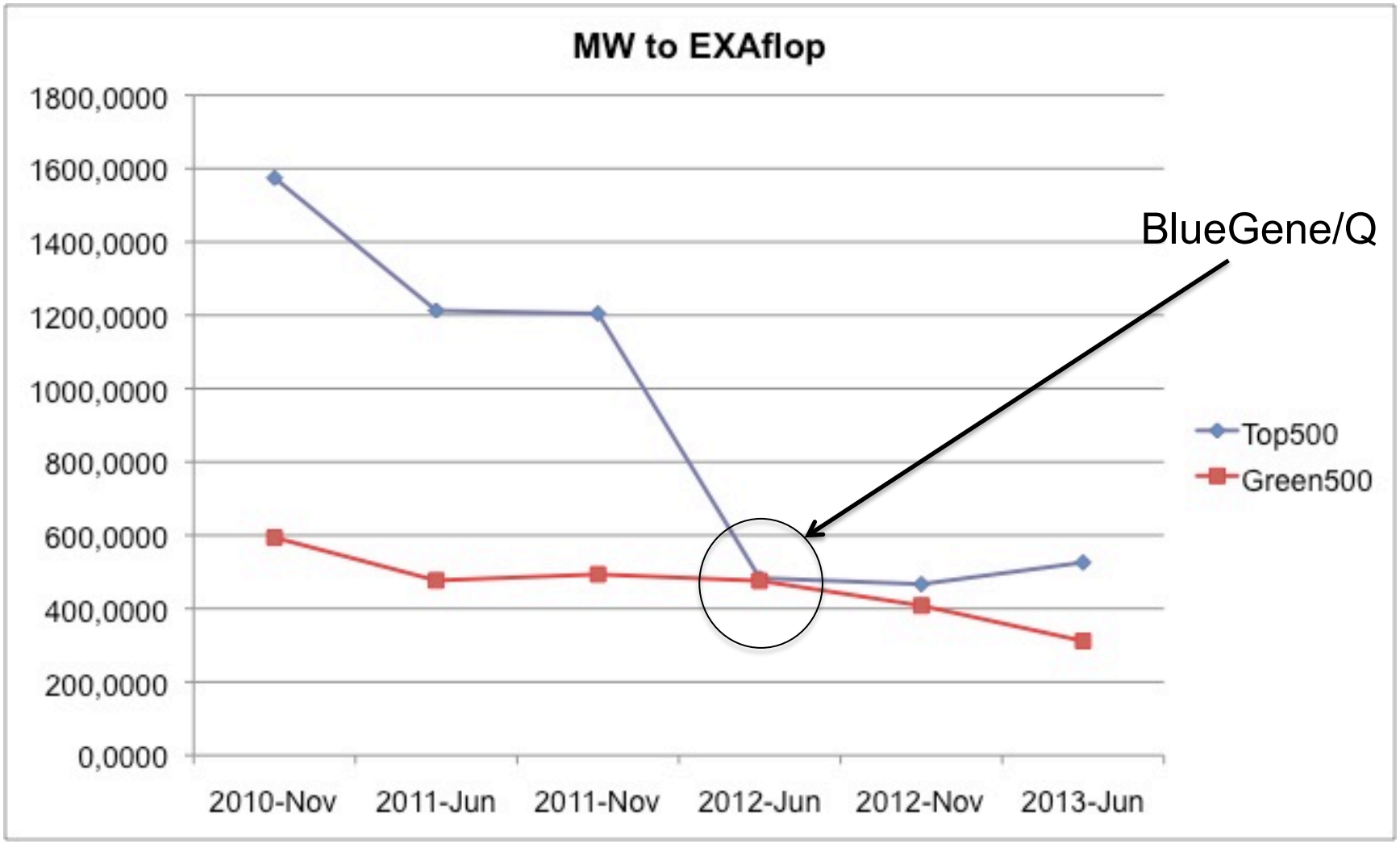
- Green500/Top500 (November 2010)

Rank Green/Top	Site, Computer	#Cores	MFLOPS/W	LINPACK (TFLOPS)	MW to EXAFLOPS?
1/115	NNSA/SC Blue Gene/Q Prototype	8.192	1.684'20	65'35	593'75
11/1	NUDT TH MPP, X5670 2.93Ghz 6C, NVIDIA GPU, FT-1000 8C	186.368	635'15	2.566'00'0	1.574'43

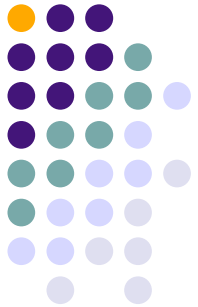
23 2,7 39 2,7

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

Concurrency and energy efficiency



Concurrency and energy efficiency



- Green500/Top500 (November 2010 – June 2013)

Nov-2010

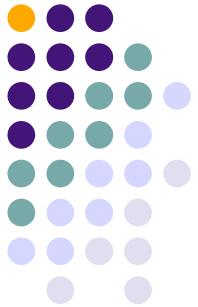
Rank Green/Top	Site, Computer	#Cores	MFLOPS/W	LINPACK (TFLOPS)	MW to EXAFLOPS?
1/115	NNSA/SC Blue Gene/Q Prototype	8.192	1.684'20	65'35	593'75
11/1	NUDT TH MPP, X5670 2.93Ghz 6C, NVIDIA GPU, FT-1000 8C	186.368	635'15	2.566'00	1.574'43

16 3 13

Jun-2013

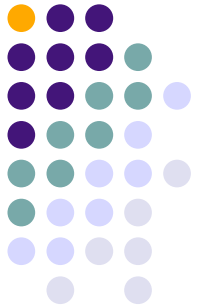
Rank Green/Top	Site, Computer	#Cores	MFLOPS/W	LINPACK (TFLOPS)	MW to EXAFLOPS?
1/467	Eurotech Aurora HPC 10-20, Xeon E5-2687W 8C 3.100GHz, Infiniband QDR, NVIDIA K20	2,688	3,208'83	98'51	311'64
20/1	Tianhe-2, TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.2GHz, TH Express-2, Intel Xeon Phi 31S1P	3,120,000	1,901'54	33,862'70	525'89

Concurrency and energy efficiency



- Reduce energy consumption is mandatory!
- High performance computing (HPC) initiatives
  - Hardware features energy saving mechanisms
  - Scientific apps are in general energy oblivious
- Mobile appliances initiatites
  - Integration of energy-saving mechanisms into embedded devices for extended battery life
- These two trends seem to be converging

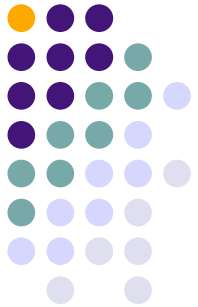
Objective



- Map of the energy-performance landscape
  - General-purpose hardware architectures
  - Specialized hardware architectures
  - Conjugate Gradient method (SPD sparse linear systems)

- Not promote one particular hardware architecture
  - Simple implementation of CG
  - Standard sparse matrix-vector product codes
  - Use of standard numerical library for the vector operations
  - Only apply compiler optimizations

Outline



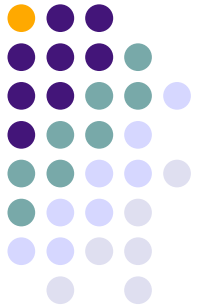
- Introduction
- Solving Sparse SPD Linear Systems (CG)
- Benchmark Matrices
- Hardware Setup and Compilers
- Experimental Results
  - Optimization with respect to time
  - Optimization with respect to the net energy
- Conclusions

Conjugate Gradient Method

$x_0 := 0$ // or any other initial guess
$r_0 := b - Ax_0$, $d_0 := r_0$
$\beta_0 := r_0^T r_0$, $\tau_0 := \| r_0 \|_2 = \sqrt{\beta_0}$
$k := 0$
while $(k < maxiter)$ & $(\tau_k > maxres)$
  $z_k := Ad_k$
  $\rho_k := \beta_k / d_k^T z_k$
  $x_{k+1} := x_k + \rho_k d_k$
  $r_{k+1} := r_k - \rho_k z_k$
  $\beta_{k+1} := r_{k+1}^T r_{k+1}$
  $\alpha_k := \beta_{k+1} / \beta_k$
  $d_{k+1} := r_{k+1} + \alpha_k d_k$
  $\tau_{k+1} := \| r_{k+1} \|_2 = \sqrt{\beta_{k+1}}$
  $k := k + 1$
end

sparse matrix-vector products
dot products → BLAS/CUBLAS
saxpy's → BLAS/CUBLAS
scalar operations

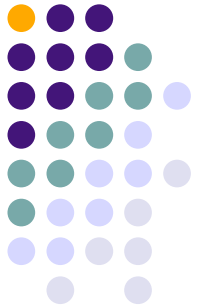
Sparse Matrix-Vector Product (CSR)



- In the Compressed Sparse Row (CSR) format, a sparse matrix is stored by using 3 vectors
- Used for multicore architectures

$$\begin{bmatrix} 5 & 2 & 4 & 0 & 0 & 2 & 0 & 5 \\ 3 & 7 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 7 & 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 2 & 0 \end{bmatrix} \xrightarrow{\text{CSR}} \begin{cases} \text{rowptr} = \begin{bmatrix} 0 & 5 & 8 & 10 & 10 & 10 & 11 & 12 & 14 \end{bmatrix} \\ \text{colind} = \begin{bmatrix} 0 & 1 & 2 & 5 & 7 & 0 & 1 & 2 & 2 & 7 & 1 & 1 & 4 & 6 \end{bmatrix} \\ \text{values} = \begin{bmatrix} 5 & 2 & 4 & 2 & 5 & 3 & 7 & 2 & 7 & 5 & 8 & 3 & 1 & 2 \end{bmatrix} \end{cases}$$

Sparse Matrix-Vector Product (CSR)

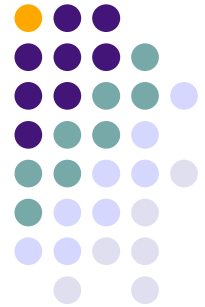


- OpenMP implementation for multicore machines

```
void spmv_csr( int n, int * rowptr, int * colind,
               float * values, float * x, float * y ) {
  int i, j;
  float dot;

  #pragma omp parallel for private ( j, dot )
  for ( i = 0; i < n; i++ ) {
    dot = 0.0;
    for ( j = rowptr [ i ]; j < rowptr [ i+1 ]; j++ )
      dot +=  values [ j ] * x [ colind [ j ] ];
    y [i] = dot ;
  }
}
```

Sparse Matrix-Vector Product (ELLPACK)

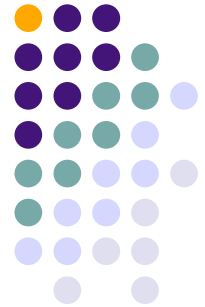


- In the ELLPACK format, a sparse matrix is stored by using 2 matrices
- Useful for streaming processors like GPU´s

$$\begin{bmatrix} 5 & 2 & 4 & 0 & 0 & 2 & 0 & 5 \\ 3 & 7 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 7 & 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 2 & 0 \end{bmatrix} \xrightarrow{\text{ELLPACK}} \text{colind} = \begin{bmatrix} 0 & 1 & 2 & 5 & 7 \\ 0 & 1 & 2 & 0 & 0 \\ 2 & 7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 4 & 6 & 0 & 0 & 0 \end{bmatrix}, \quad \text{values} = \begin{bmatrix} 5 & 2 & 4 & 2 & 5 \\ 3 & 7 & 2 & 0 & 0 \\ 7 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 \end{bmatrix}$$

Overhead (65%)

Sparse Matrix-Vector Product (ELLPACK)

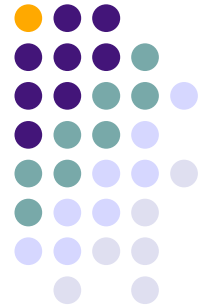


- CUDA implementation for GPU's (nVidia)

```
__global__ void spmv_ell( int n, int max_nzr, int * colind,
                          float * values, float * x, float * y ) {
  int i, j, row, col;
  float dot;

  row = blockDim.x * blockIdx.x + threadIdx.x;
  if ( row < n ){
    dot = 0.0;
    for ( j = 0; j < max_nzr; j++ ) {
      col = colind [ n * j + row ];
      if ( values [ n * j + row ] != 0 )
        dot += values [ n * j + row ] * x [ col ];
    }
    y [ row ] = dot;
  }
}
```

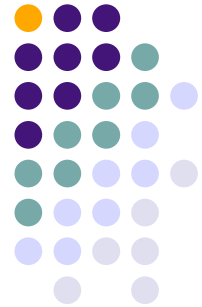
Benchmark Matrices



- Main features of the sparse matrices
  - 1 discretization of a Laplacian equation in a 3D unit cube
  - 6 from the University of Florida Sparse Matrix Collection

Source	Acronym	Matrix	#nonzeros (nz)	Size (n)	nz/n
Laplace	A159	A159	27986067	4019679	6,96
UFMC	audi	audikw_1	77651847	943695	82,28
	bmw	bmwcra_1	10644002	148770	71,55
	crank	crankseg_2	14148858	63838	221,64
	f1	F1	26837113	343791	78,06
	inline	inline_1	36816342	503712	73,09
	ldoor	ldoor	46522475	952203	48,86

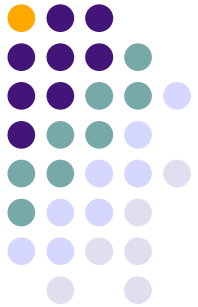
Benchmark Matrices



- Overhead of the sparse matrices
  - We have to compare the ratio and the maximum of the nonzeros elements in the rows of the matrix

Source	Acronym	Matrix	nz/n	max(nz_row)	overhead
Laplace	A159	A159	6,96	7	0,54%
UFMC	audi	audikw_1	82,28	345	76,15%
	bmw	bmwcra_1	71,55	351	79,62%
	crank	crankseg_2	221,64	3423	93,53%
	f1	F1	78,06	435	82,05%
	inline	inline_1	73,09	843	91,33%
	ldoor	ldoor	48,86	77	36,55%

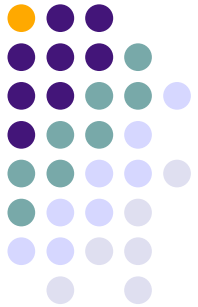
Hardware Setup and Compilers



- Main features of the general-purpose multicores

Acron.	Architecture	#cores	RAM size, type	Compiler
AIL	AMD Opteron 6276 (Interlagos)	8	64GB, DDR3 1.3GHz	gcc 4.4.6
AMC	AMD Opteron 6128 (Magny-Cours)	8	48GB, DDR3 1.3GHz	gcc 4.4.6
INH	Intel Xeon E5504 (Nehalem)	8	32GB, DDR3 800MHz	gcc 4.1.2
ISB	Intel E5-2620 (Sandy-Bridge)	6	32GB, DDR3 1.3GHz	gcc 4.1.2

Hardware Setup and Compilers



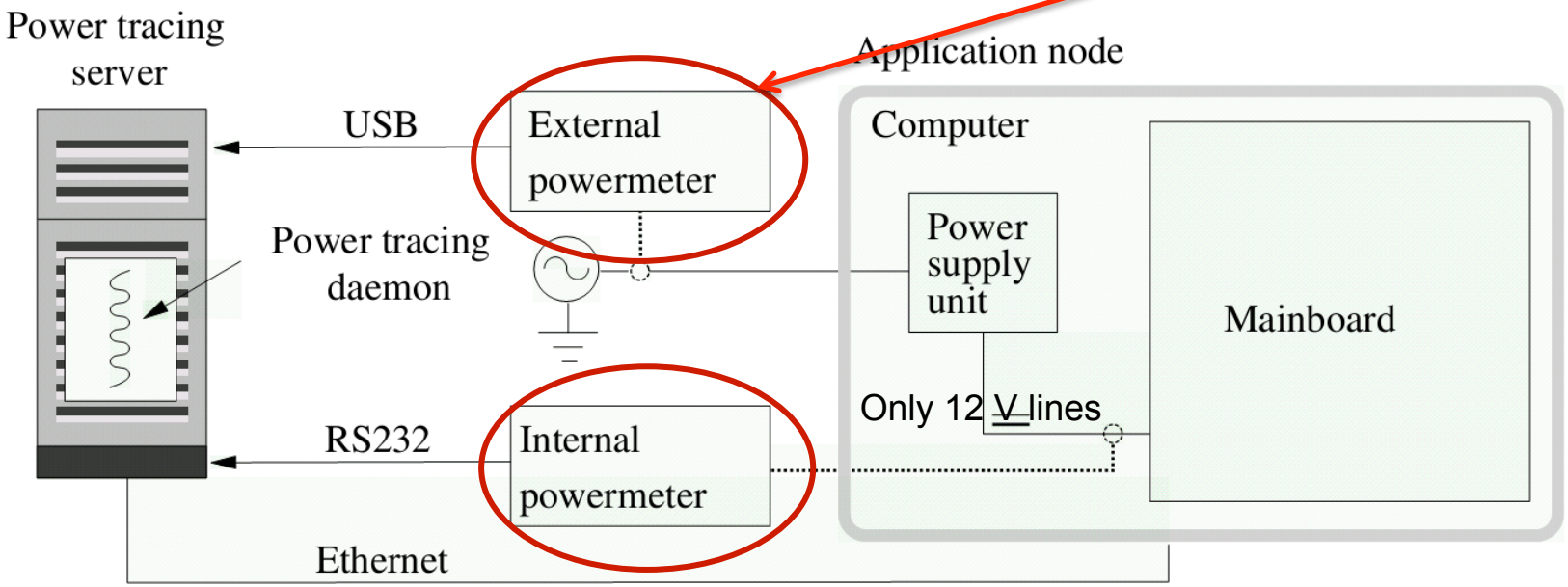
- Main features of the the two low-power multicores, a low-power digital signal multicore (DSP) and three GPU´s

Acron.	Architecture	#cores	RAM size, type	Compiler
IAT	Intel Atom D510	2	1GB, DDR2 533MHz	gcc 4.5.2
ARM	ARM Cortex A9	4	2GB, DDR3L	gcc 4.5.2
TIC	Texas Instruments C6678	8	512MB, DDR3	cl6x 7.4.1

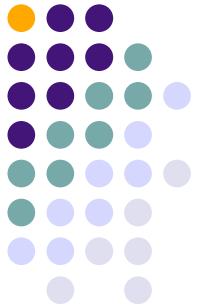
Acron.	Architecture	#cores	RAM size, type	Compiler
QDR	ARM Cortex A9 NVIDIA Quadro 1000M	4 96	2GB, DDR3L 2GB, DDR3	gcc 4.5.2 nvcc 4.2
FER	Intel Xeon E5520 NVIDIA Tesla C2050 (Fermi)	8 448	24GB, 3GB, GDDR5	gcc 4.4.6 nvcc 4.2
KEP	Intel Xeon i7-3930K NVIDIA Tesla K20 (Kepler)	6 2,496	24GB, 5GB, GDDR5	gcc 4.4.6 nvcc 4.2

Measurement of the consumption

- WattsUp?Pro powermeter:
  - External powermeter connected to the power supply
  - Sampling freq. = 1 Hz and accuracy of ±1’5%
  - USB connection

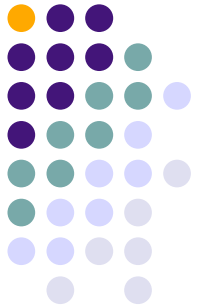


Measurement of the consumption



- WattsUp?Pro powermeter
- When the codes are working,
  - Total energy is the consumption of all the machine
  - Net energy is the consumption of the processor
- To compute net energy, we need to know the consumption of the other devices,
  - Measuring the consumption when the system is idle
  - Subtracting the idle energy to the total energy

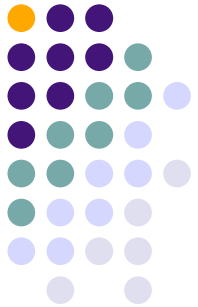
Hardware, Frequencies and Idle Power



- Frequency and Idle power of the general-purpose multicores

Acron.	Architecture	#cores	Frequency (GHz) -- Idle power (W)
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
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
INH	Intel Xeon E5504 (Nehalem)	8	2.0 -- 280.6, 2.33 -- 281.48, 2.83 -- 282.17
ISB	Intel E5-2620 (Sandy-Bridge)	6	1.2 -- 93.35, 1.4 -- 93.51, 1.6 -- 93.69, 1.8 -- 93.72, 2.0 -- 93.5

Hardware, Frequencies and Idle Power

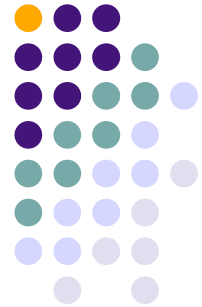


- Frequency and Idle power of the two low-power multicores, a low-power digital signal multicore (DSP) and three GPU´s

Acron.	Architecture	#cores	Frequency (GHz) -- Idle power (W)
IAT	Intel Atom D510	2	0.8 -- 11.82, 1.06 -- 11.59, 1.33-11.51, 1.6 -- 11.64
ARM	ARM Cortex A9	4	0.62 -- 11.7, 1.3 -- 12.2
TIC	Texas Instruments C6678	8	1.0 -- 18.0

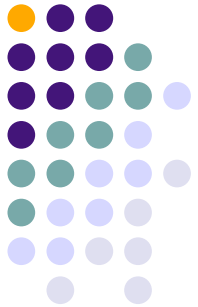
Acron.	Architecture	#cores	Frequency (GHz) -- Idle power (W)
QDR	ARM Cortex A9 NVIDIA Quadro 1000M	4 96	0.120 -- 11.2, 1.3 -- 12.2 1.4
FER	Intel Xeon E5520 NVIDIA Tesla C2050 (Fermi)	8 448	1.6 -- 222.0, 2.27 -- 226.0 1.15
KEP	Intel Xeon i7-3930K NVIDIA Tesla K20 (Kepler)	6 2,496	1.2 -- 106.30, 3.2 -- 106.50 0.7

Experimental Results



- The compilation
  - Optimization flag -O3 in all cases
- The execution of CG
  - All the entries of the solution are equal to 1.
  - Tests on all available frequencies and #cores
  - Considering the time and the power consumption.
- Power measurement
  - The average power while idle for 30 seconds
  - Warm up period of 5 minutes
  - All tests were executed for a minimum of 3 minutes

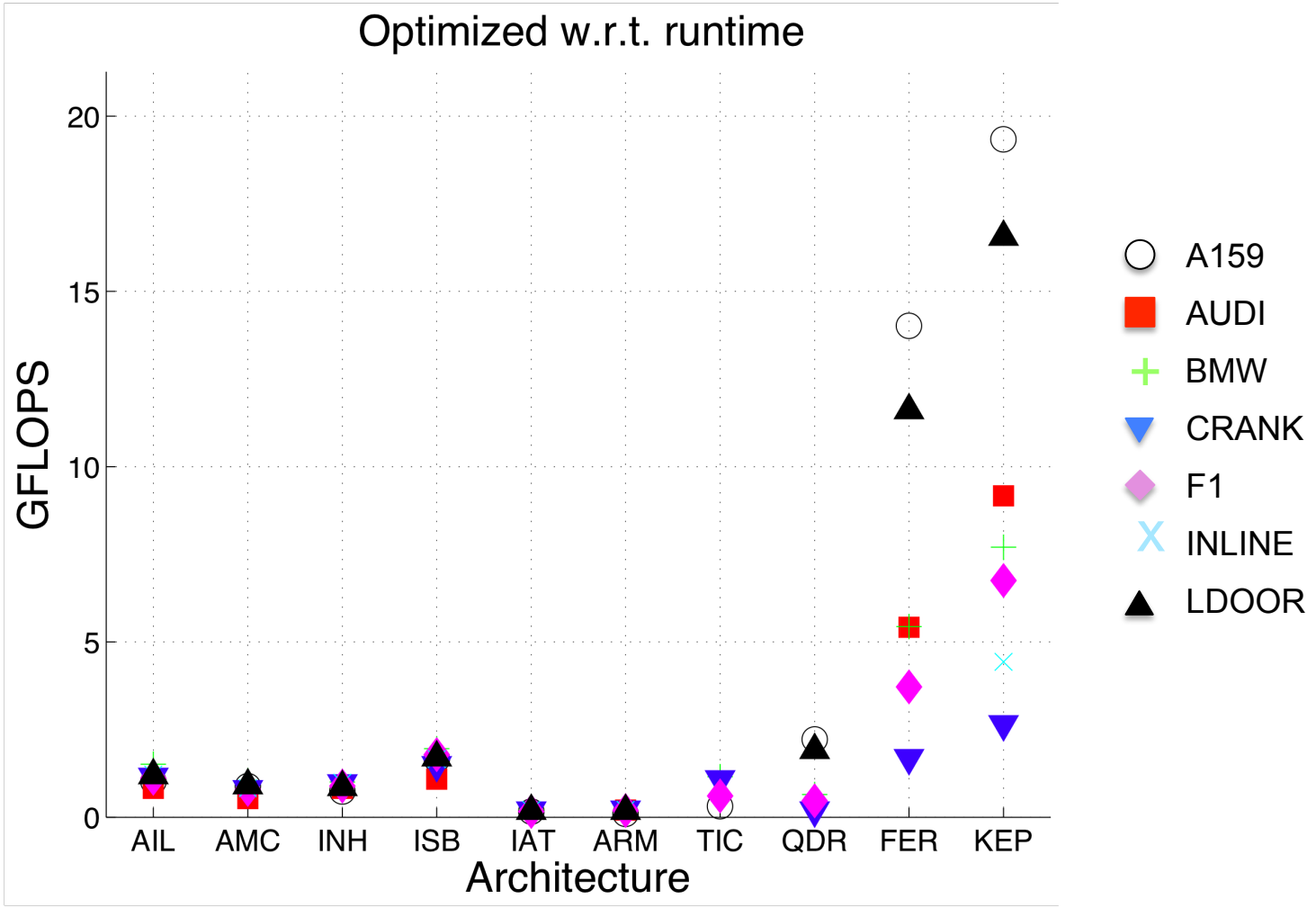
Experimental Results



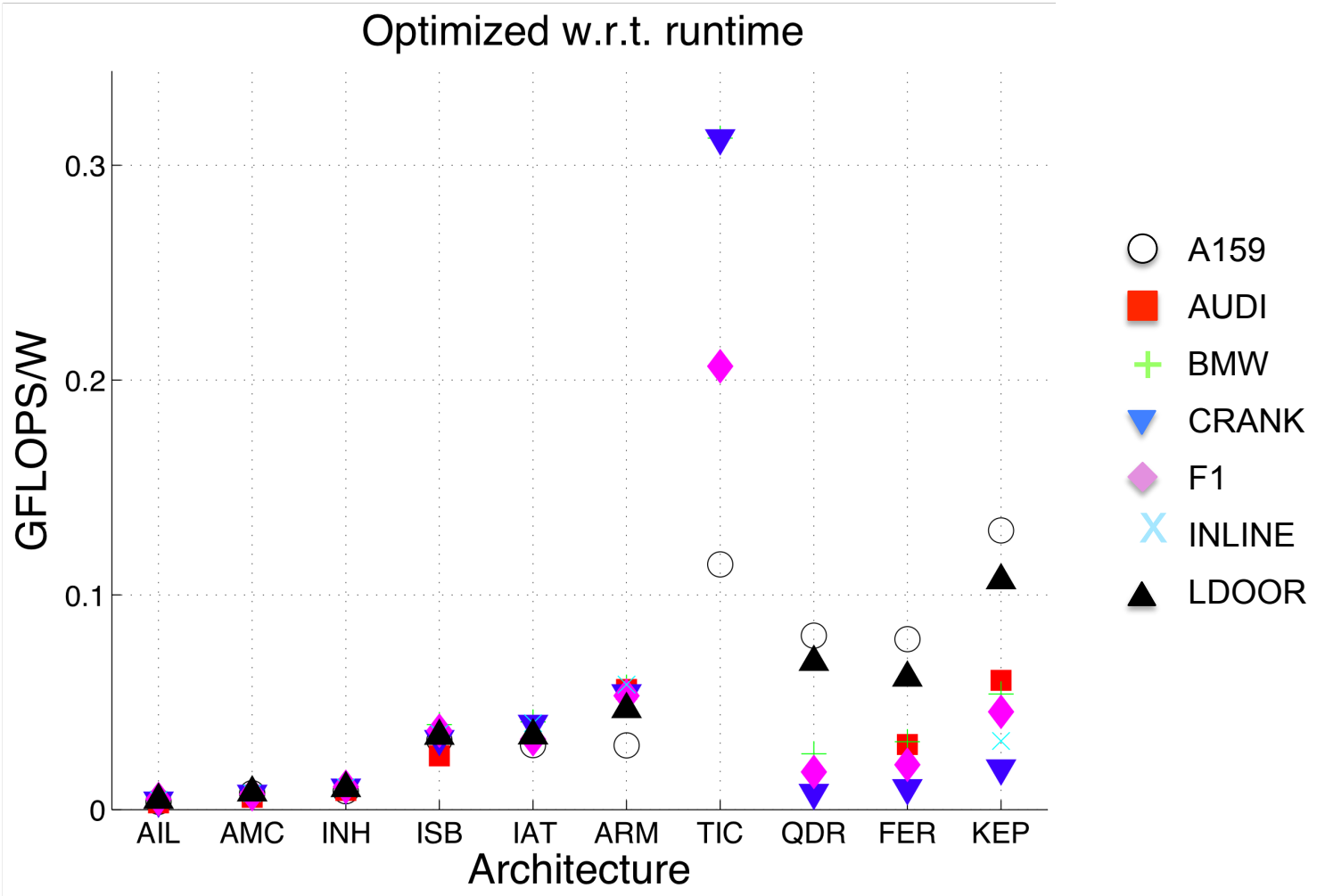
- We have summarized the results:
  - The best case from the perspectives of time and net energy
  - We show the number of cores, the frequency, the execution time, the net energy and the total energy

AIL										
	Optimized w.r.t time					Optimized w.r.t net energy				
Matrix	#cores	freq.	exec. time	net energy	total energy	#cores	freq.	exec. time	net energy	total energy
A159	8	2300	0,08	20,60	33,30	8	2100	0,08	12,40	25,20
AUDI	8	2300	0,19	51,60	84,40	8	2100	0,28	42,90	89,70
BMW	8	2300	0,01	3,92	6,38	8	2100	0,02	3,05	6,27
CRANK	8	2300	0,02	6,66	10,80	8	2100	0,04	5,52	11,50
F1	6	2300	0,05	11,30	19,80	6	2100	0,07	9,82	21,20
INLINE	8	2300	0,07	17,60	28,70	8	2100	0,09	14,30	29,60
LDOOR	8	2300	0,08	19,90	32,60	8	2100	0,09	14,30	29,60

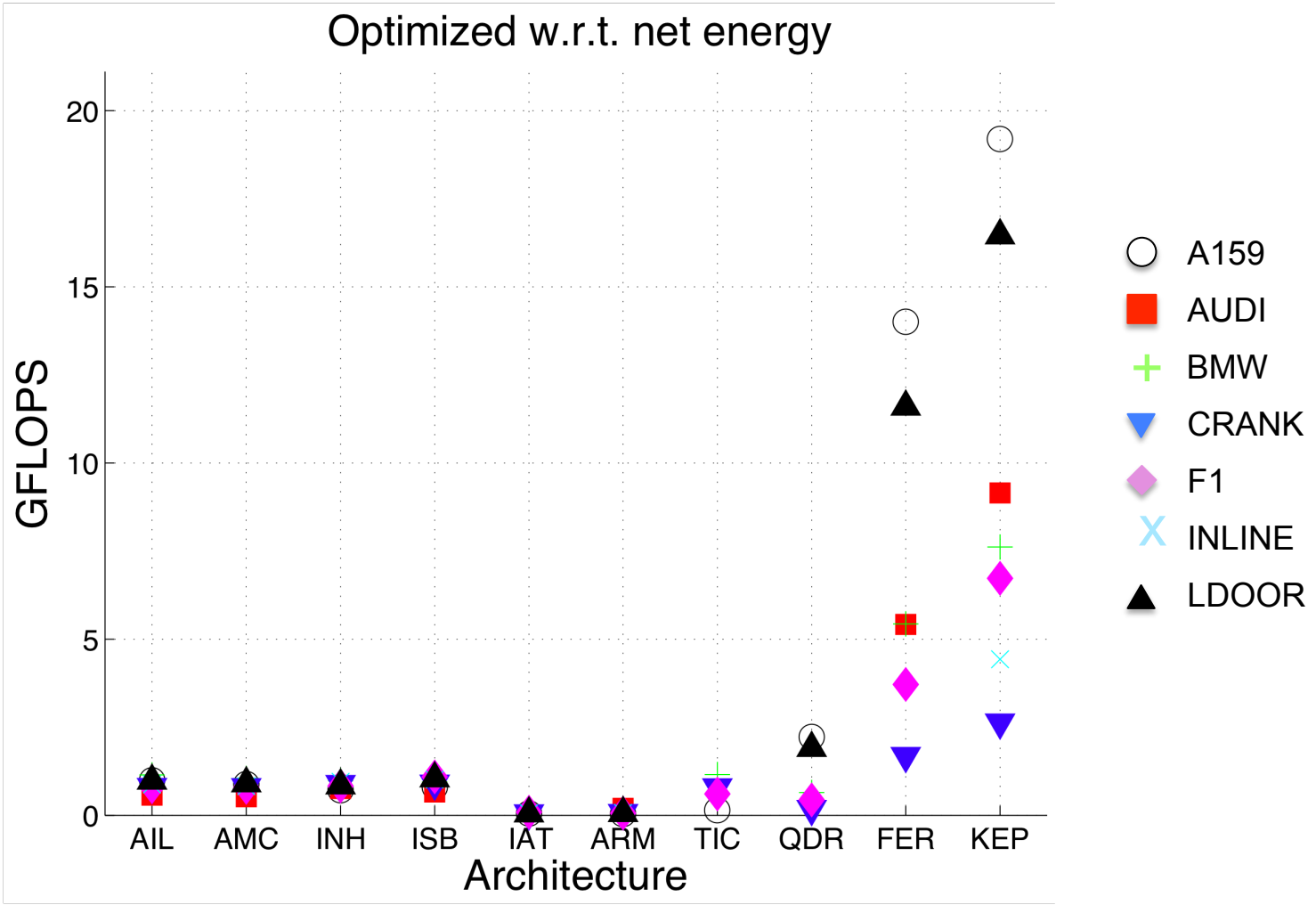
Experimental Results



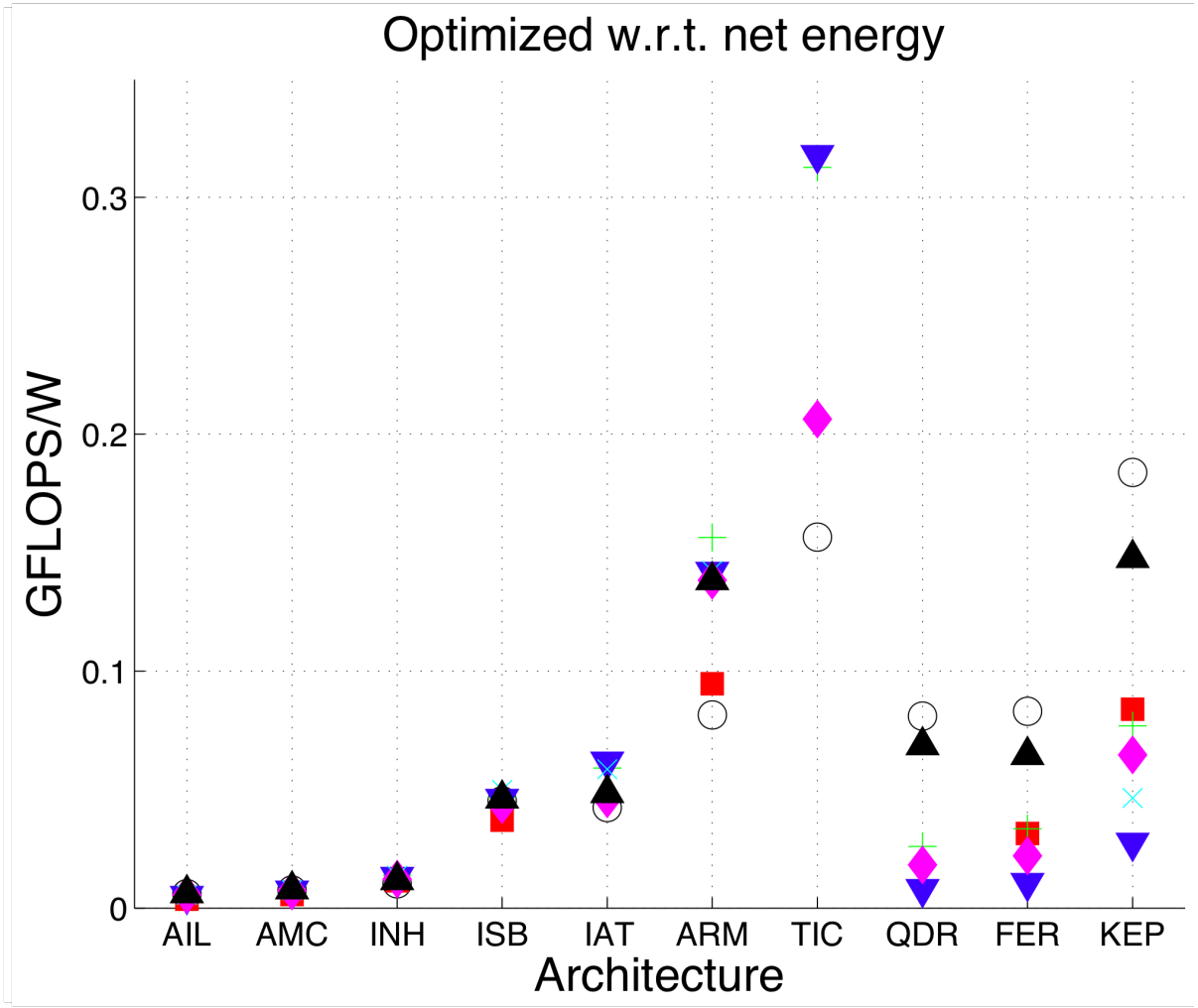
Experimental Results



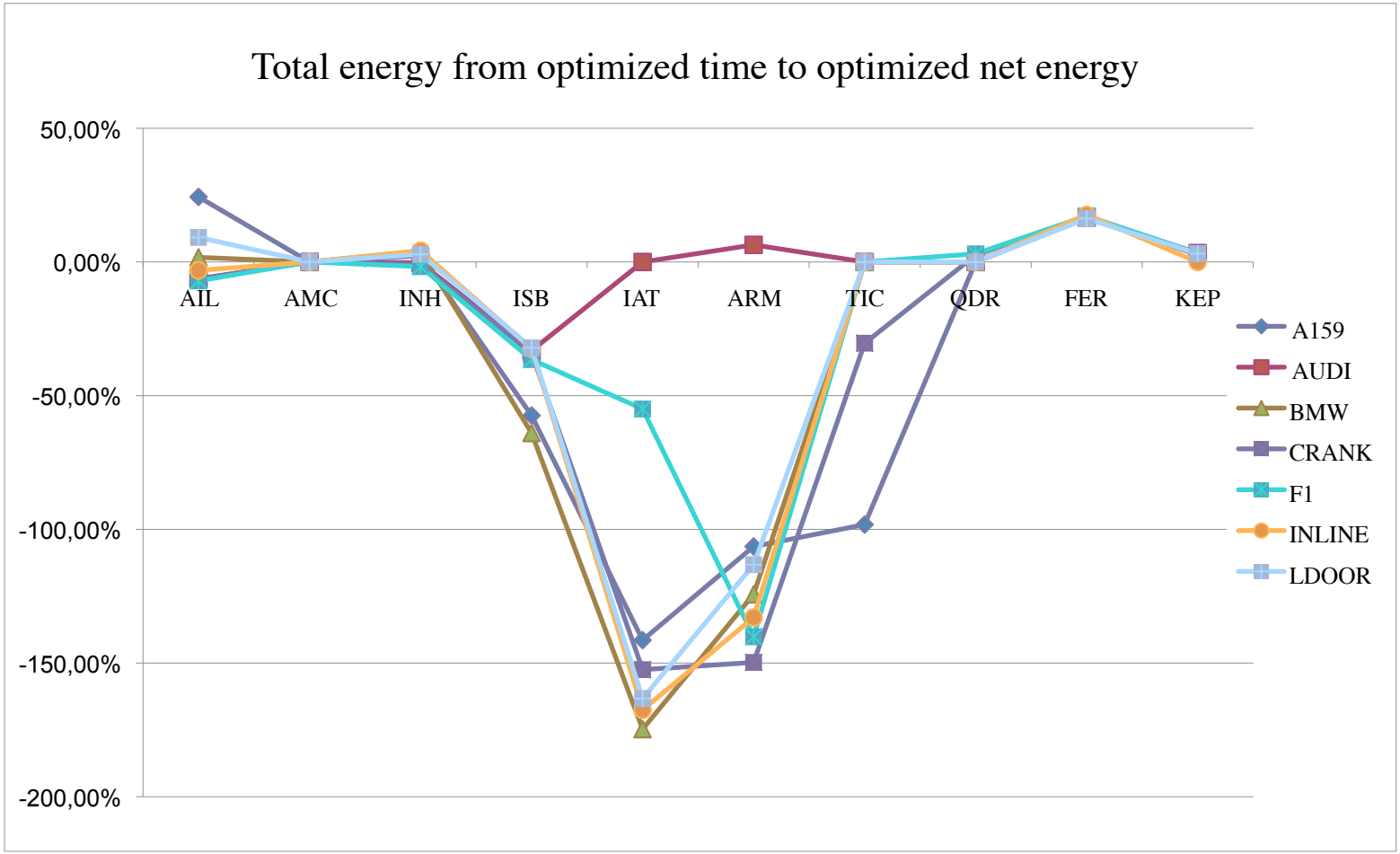
Experimental Results



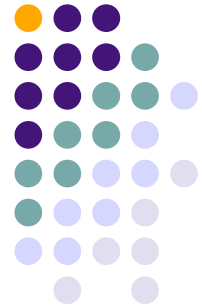
Experimental Results



Experimental Results



Experimental Results

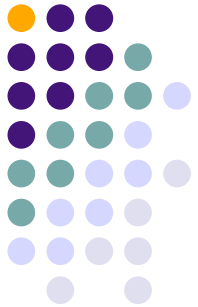


- Net energy vs total energy:

$$E_{net} = P_{net} * T \quad , \quad E_{tot} = \left(P_{idle} + P_{net}\right) * T = E_{net} + P_{idle} * T$$

ARM						
	Optimized w.r.t time			Optimized w.r.t net energy		
Matrix	exec. time	net energy	total energy	exec. time	net energy	total energy
A159	0,82	2,54	12,50	2,12	0,98	25,80
AUDI	0,79	2,86	12,50	0,82	1,69	11,70
BMW	0,11	0,39	1,73	0,32	0,14	3,88
CRANK	0,13	0,53	2,05	0,42	0,20	5,12
F1	0,29	1,04	4,58	0,91	0,40	11,00
INLINE	0,38	1,38	6,01	1,15	0,56	14,00
LDOOR	0,50	1,92	8,02	1,41	0,65	17,10

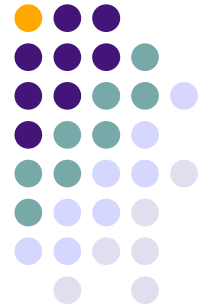
Conclusions & Future work



- Evaluation of the CG on a variety of architectures:
  - The GPU's obtain the largest performances
  - The DSP obtain better energy efficiency than the GPU´s
  - The ARM is also competitive with the GPU's
  - Good energy results for the Intel Atom and Sandy-bridge
  - We need to consider the net energy and the total energy.

- In the next future, we will analyze other problems:
  - sorting algorithms or image processing

Performance and Energy Analysis of the Iterative Solution of Sparse Linear Systems on Multicore and Manycore Architectures



Thanks for your attention!

Any question?