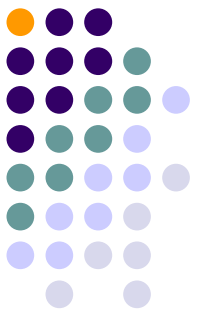


Energy-Aware Matrix Computations on Multi-Core and Many-core Platforms

Enrique S. Quintana-Ortí



Performance and energy consumption

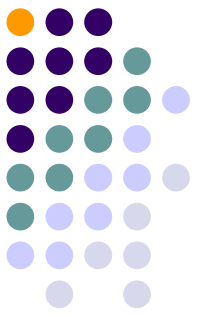


- Top500 (November 2011)

Rank	Site	#Cores	LINPACK (TFLOPS)
1	RIKEN AICS K Computer– Spar64 V8ifx (8-core)	705,024	10,510.00*
2	NSC Tianjin – NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050	186,368	2,566.00
3	DOE ORNL – Cray XT5-HE Opteron 6-core 2.6 GHz	224,162	1,759.00
9	CEA (France) – Bull bullx super-node S6010/S6030	138,368	1,050.00
114	BSC (Spain) – Bull B505, Xeon E5649 6C 2.53 GHz, NVIDIA 2090	5,544	103.20

* 1 day K Computer = 394 years of the world population (7.000 million people) with a hand calculator

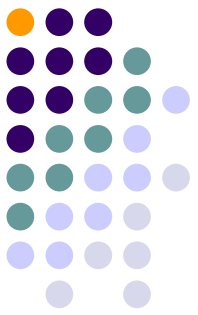
Performance and energy consumption



- Green500 (November 2011)

Rank Green/Top	Site	#Cores	MFLOPS/W	LINPACK (TFLOPS)
1/29	IBM Rochester – BlueGene/Q, Power BQC 16C 1.60 GHz	32,768	2,026.48	339.83
7/114	BSC (Spain) – Bull B505, Xeon E5649 6C 2.53 GHz, NVIDIA 2090	5,544	1,266.26	103.20
32/1	RIKEN AICS K Computer– Spar64 Vlllfx (8-core)	705,024	830.18	10,510.00
47/2	NSC Tianjin – NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050	186,368	635.15	2,566.00
53/3	DOE ORNL – Cray XT5-HE Opteron 6-core 2.6 GHz	582,00	Cray	1,759.00

Multi-core and many-core platforms



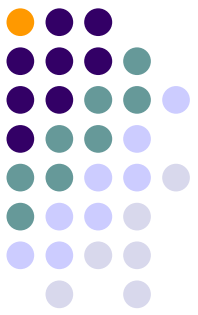
- “Conventional” architectures



- New challengers...



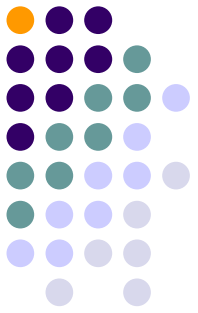
Matrix computations



- Linear algebra? Please, don't run away!
 - Determinants, linear systems, least squares fitting, FFT, etc.
- Importance:
 - Intel MKL, AMD ACML, IBM ESSL, NVIDIA CUBLAS, ongoing for TI

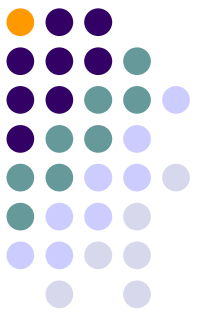


Index



1. Scientific applications
2. Leveraging concurrency
3. Cost of energy

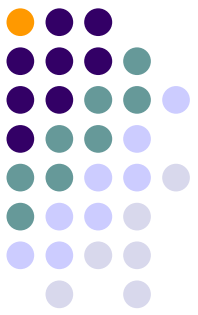
Index



1. Scientific applications
2. Leveraging concurrency
3. Cost of energy

Scientific applications

Biological systems



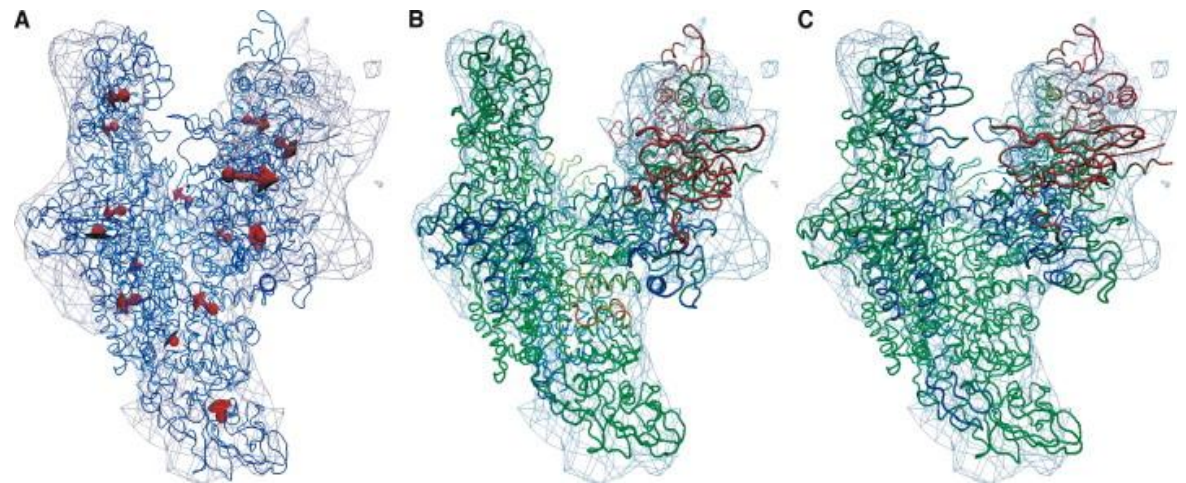
- Simulations of molecular dynamics

- Solve

$$AX = BX\Lambda,$$

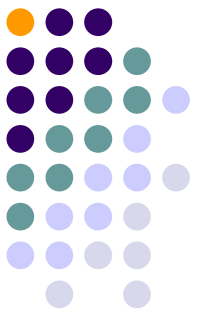
dense $A, B \rightarrow n \times n$

$n = 134,484$



Scientific applications

Industrial processes



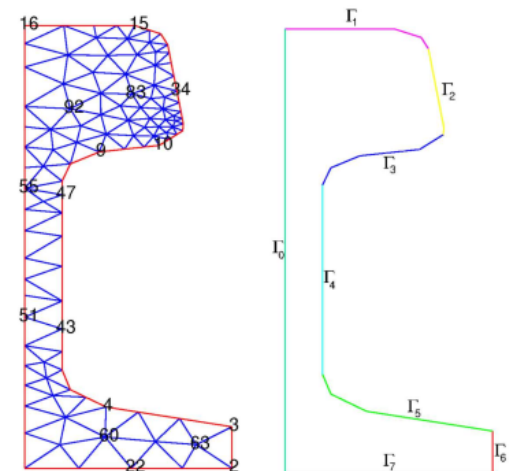
- Optimal cooling of steel profiles

- Solve

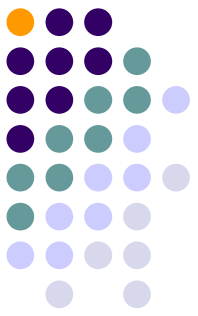
$$A^T X + X A - X S X + Q = 0,$$

dense $A \rightarrow n \times n$

$n = 5,177$ for a mesh
width of $6.91 \cdot 10^{-3}$

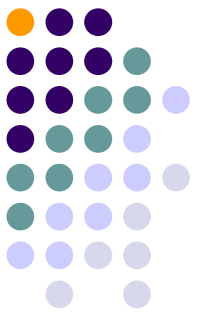


Scientific applications Summary



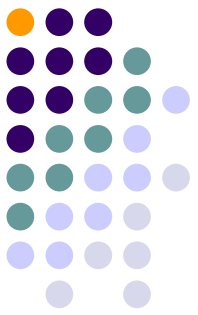
- Dense linear algebra is at the bottom of the “food chain” for many scientific and engineering apps.
 - Fast acoustic scattering problems
 - Dielectric polarization of nanostructures
 - Magneto-hydrodynamics
 - Macro-economics

Index



1. Scientific applications
2. Leveraging hardware concurrency
3. Cost of energy

Leveraging hw. concurrency Threads



- Linear system

$$\left. \begin{array}{l} 2x + 3y = 3 \\ 4x - 5y = 6 \end{array} \right\}$$

$A X = B$, with dense A, B

$\rightarrow n \times n$:

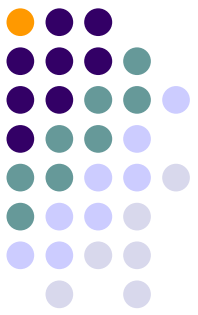
$\approx 2n^3/3 + 2n^3$ flops

- Intel Xeon:

4 DP flops/cycle, e.g.,
at $f=2.0$ GHz

n	Time 1 core	Time 8 cores	Time 16-node cluster, 8-core per node, i.e., 192 cores
100	33.33 ms	--	--
1.000	0.33 s	--	--
10^4	333.33 s	41.62 s	--
10^5	> 92 h	> 11 h	> 28 m

Leveraging hw. concurrency Threads



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

2010 JUGENE

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

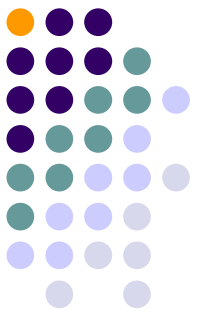


2020 EFLOPS (10^{18} flops/sec.)

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

Leveraging hw. concurrency

Cholesky factorization



$$A = L * L^T$$

Key in the solution of s.p.d. linear systems

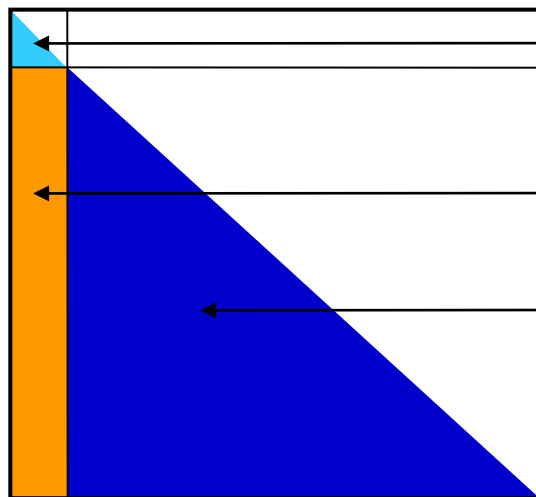
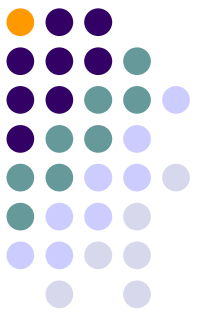
$$A x = b \equiv (LL^T) x = b$$

$$L y = b \Rightarrow y$$

$$L^T x = y \Rightarrow x$$

Leveraging hw. concurrency

Cholesky factorization (blocked)



1st iteration

F: $A_{11} = L_{11} * L_{11}^T$

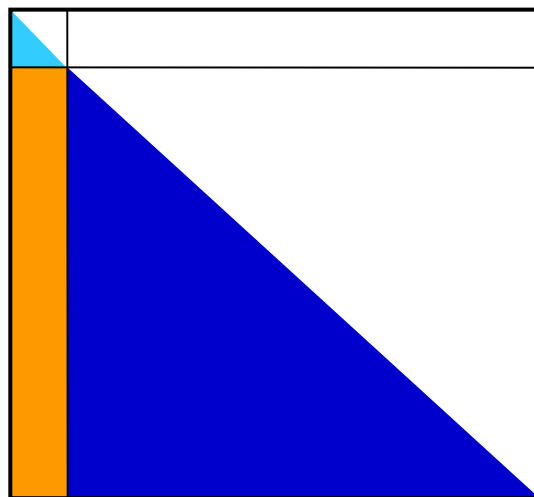
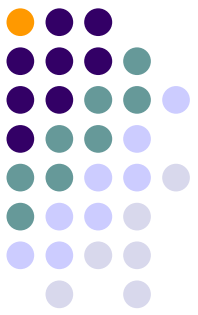
T: $L_{21} \leftarrow A_{21} * L_{11}^{-T}$

U: $A_{22} \leftarrow A_{22} - L_{21} * L_{21}^T$

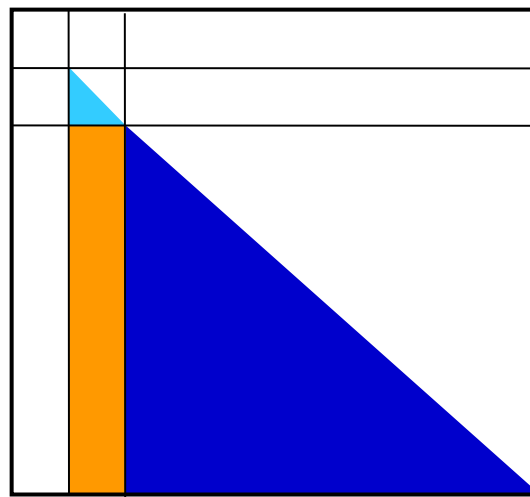
- Reuse data in cache
- MT processor: Employ a MT implementation of T and P

Leveraging hw. concurrency

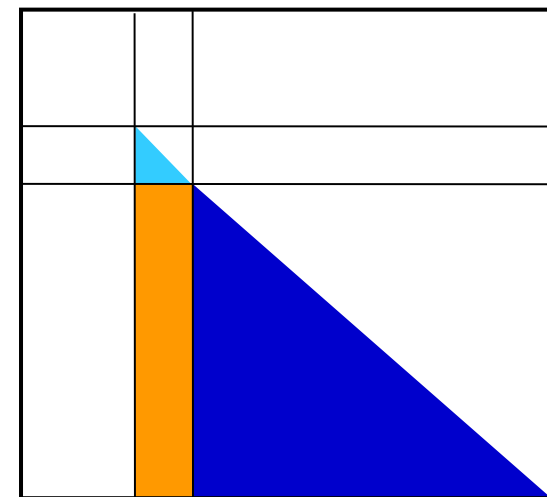
Cholesky factorization (blocked)



1st iteration



2nd iteration

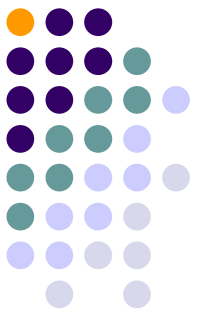


3rd iteration

...

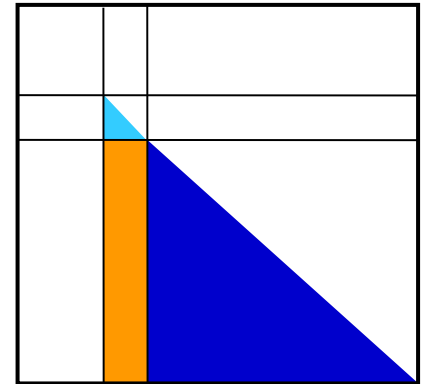
Leveraging hw. concurrency

Cholesky factorization (blocked)



```
for (k=1; k<=n/b; k++){
```

F: $\text{Chol}(A[k,k]); \quad // \quad A_{kk} = L_{kk} * L_{kk}^T$



```
if (k<=n/b) {
```

T: $\text{Trsm}(A[k,k], A[k+1,k]); \quad // \quad L_{k+1,k} \leftarrow A_{k+1,k} * L_{kk}^{-T}$

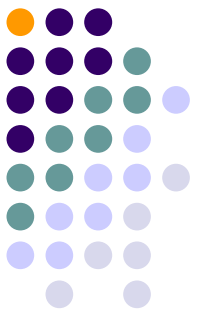
U: $\text{Syrk}(A[k+1,k], A[k+1,k+1]); \quad // \quad A_{k+1,k+1} \leftarrow A_{k+1,k+1} - L_{k+1,k} * L_{k+1,k}^T$

```
}
```

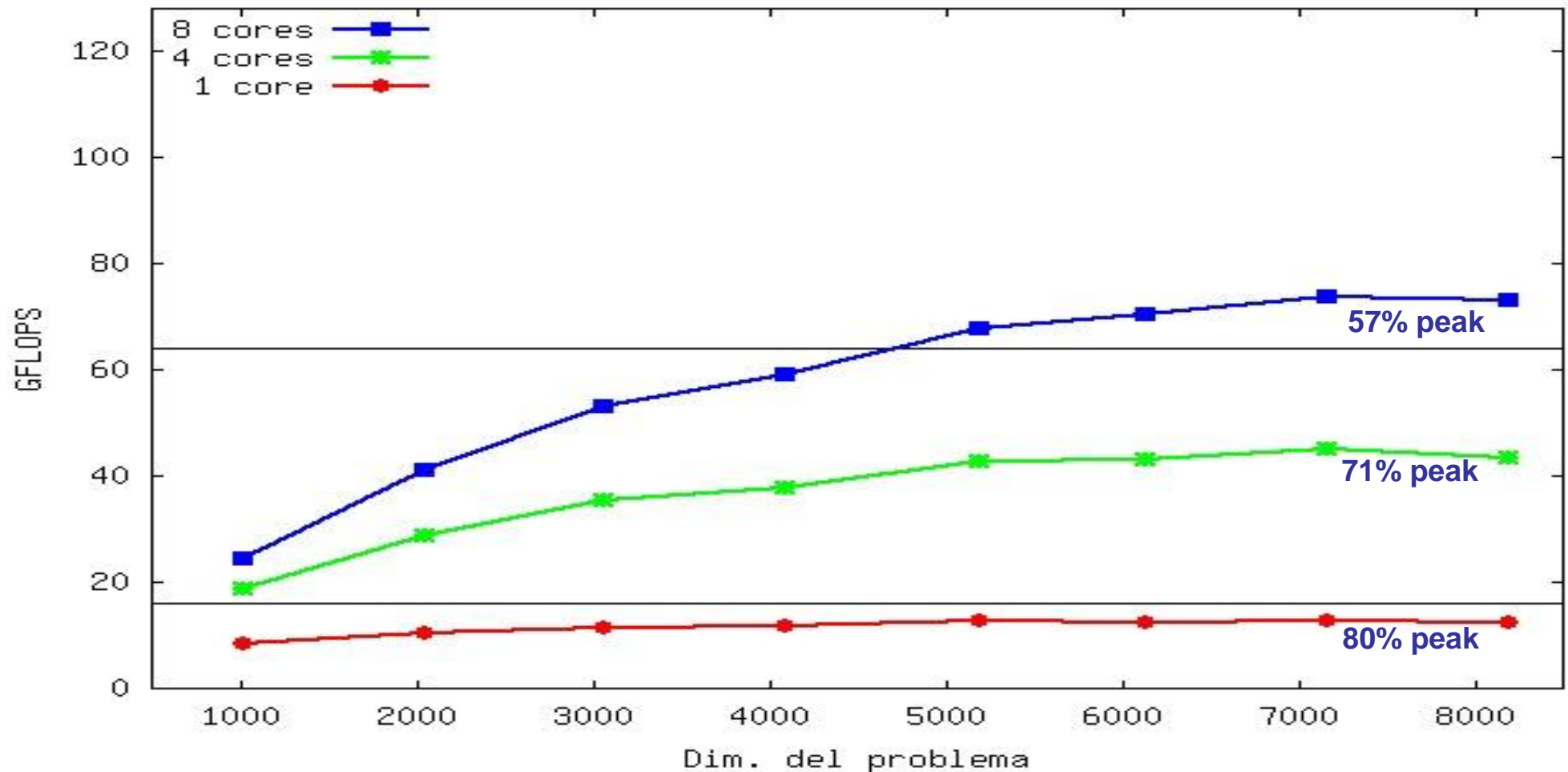
```
}
```

Leveraging hw. concurrency

Cholesky factorization (blocked)

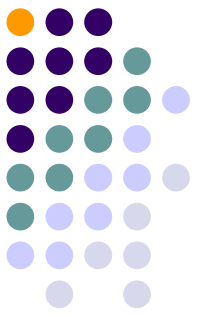


Factor de Cholesky en 2 Intel Xeon QuadCore (8 cores)



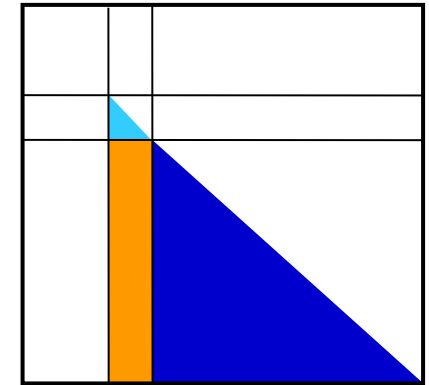
Leveraging hw. concurrency

Algorithmic parallelism



Why?

Excessive thread synchronization



```

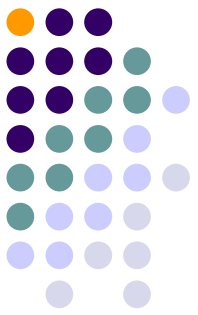
for (k=1; k<=n/b; k++) {
F:   Chol (A[k,k]) ;                               //  $A_{kk}$           =  $L_{kk} * L_{kk}^T$ 

      if (k<=n/b) {
T:   Trsm (A[k,k] , A[k+1,k]) ;                     //  $L_{k+1,k}$        $\leftarrow A_{k+1,k} * L_{kk}^{-T}$ 

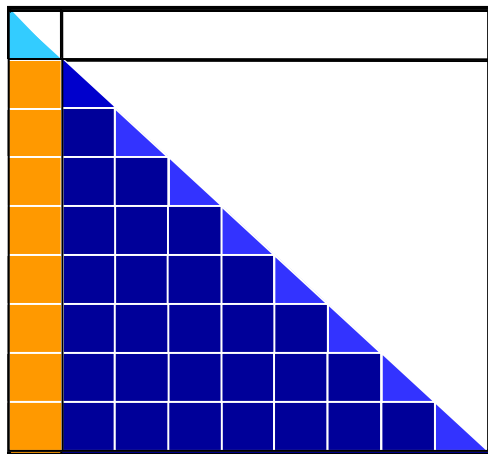
U:   Syrk (A[k+1,k] , A[k+1,k+1]) ; //  $A_{k+1,k+1}$   $\leftarrow A_{k+1,k+1}$ 
                                          //  $- L_{k+1,k} * L_{k+1,k}^T$ 
      }
}
    
```

Leveraging hw. concurrency

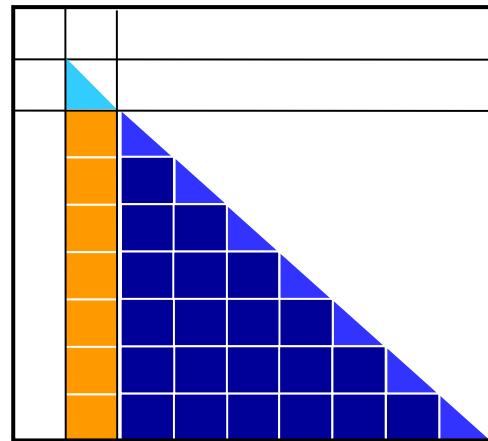
Algorithmic parallelism



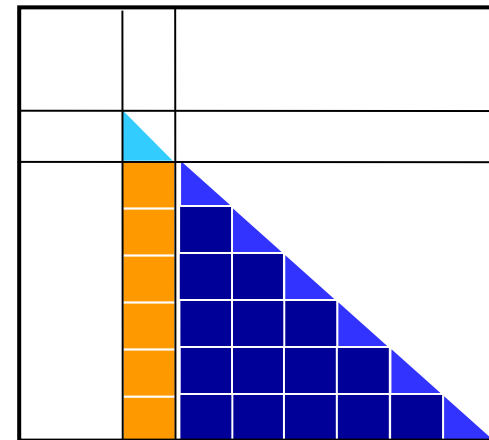
- ...but there is much more parallelism!!!



1st iteration



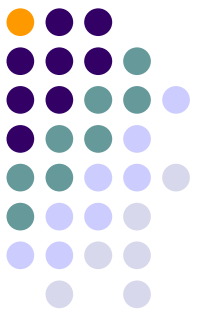
2nd iteration



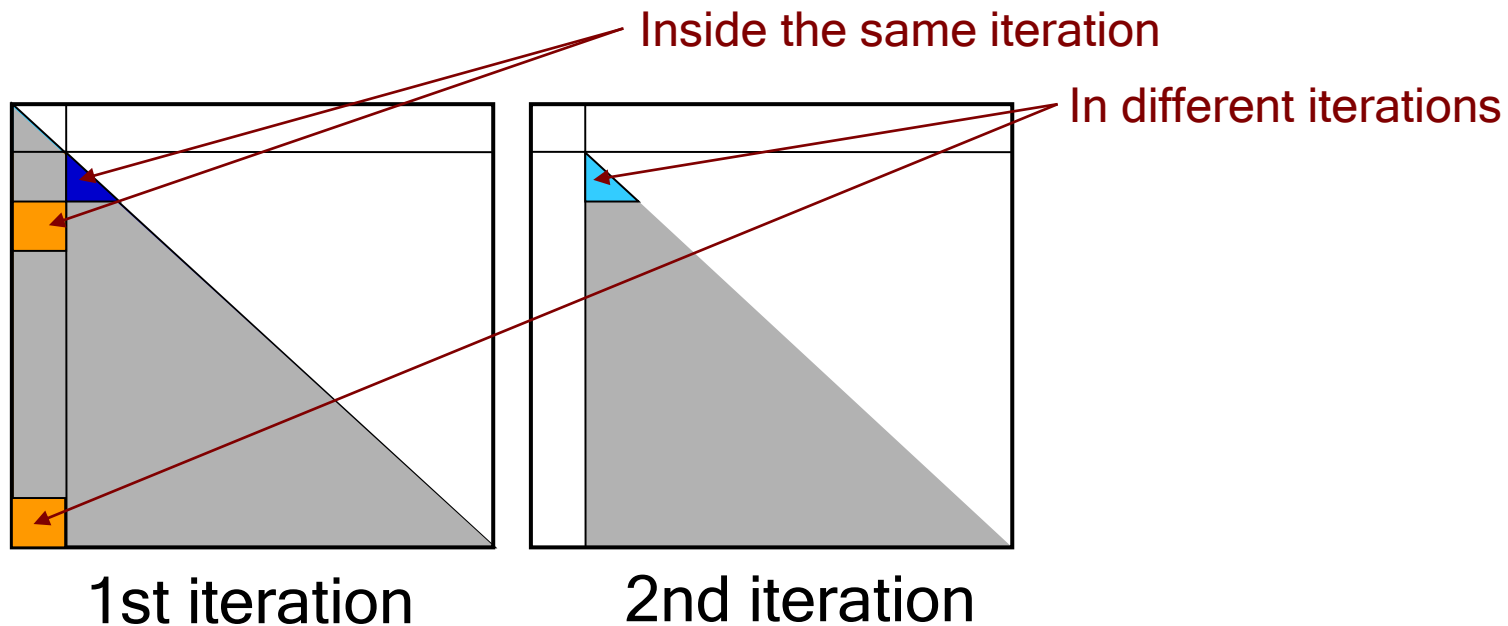
3rd iteration

Leveraging hw. concurrency

Algorithmic parallelism



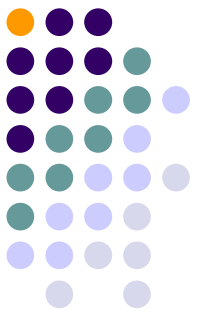
- ...but there is much more parallelism!!!



How can we leverage it?

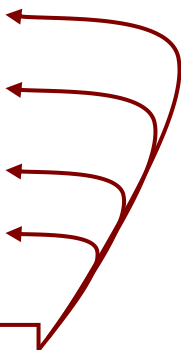
Leveraging hw. concurrency

Task parallelism

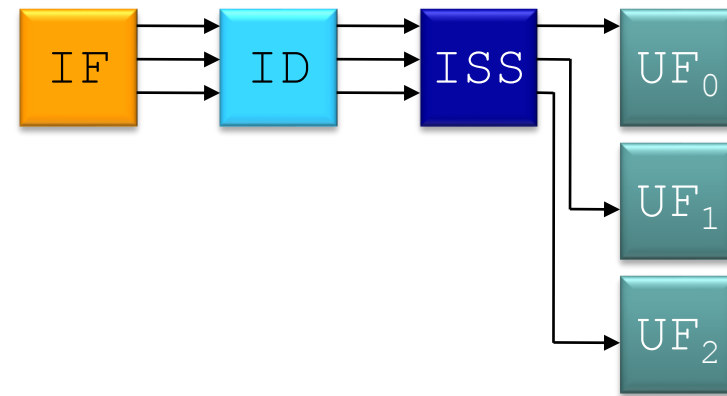


Scalar code

```
loop: ld    f0, 0(r1)
      addd  f4, f0, f2
      sd    f4, 0(r1)
      addi  r1, r1, #8
      subi  r2, r2, #1
      bnez  r2, loop
```

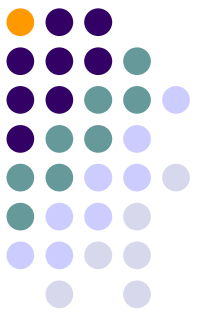


(Super)scalar processor

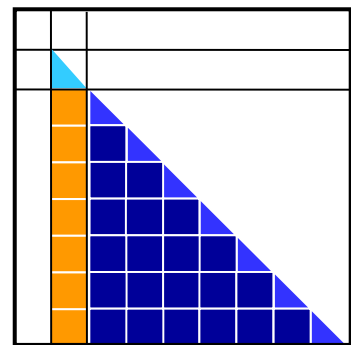
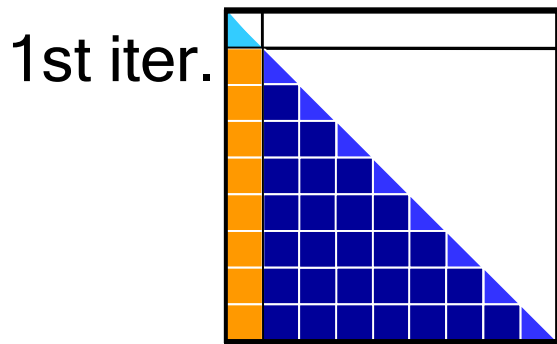


Leveraging hw. concurrency

Task parallelism

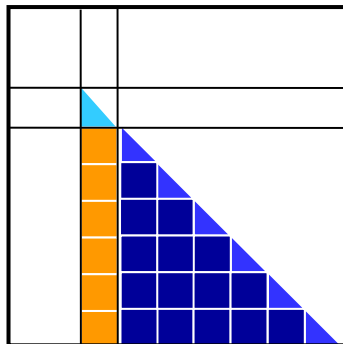


- Something similar for (dense) linear algebra?



2nd iter.

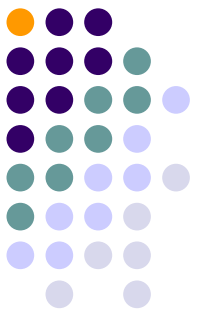
3rd iter.



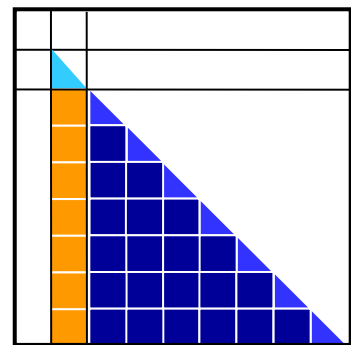
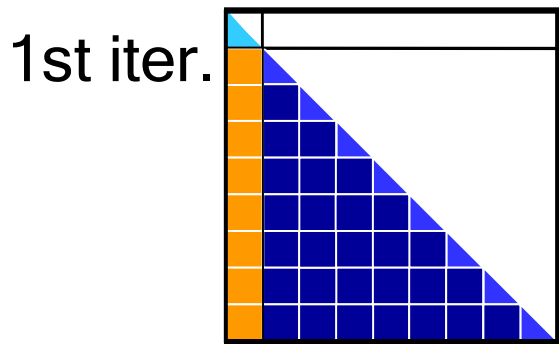
```
for (k=1; k<=n/b; k++){  
  F: Chol(A[k,k]);  
  for (i=k+1; i<=n/b; i++){  
    T: Trsm(A[k,k], A[i,k]);  
    U1: Syrk(A[i,k], A[i,i]);  
    for (j=k+1; j<=i; j++){  
      U2: Gemm(A[i,k], A[j,k], A[i,j]);  
    }  
  }  
}
```

Leveraging hw. concurrency

Task parallelism

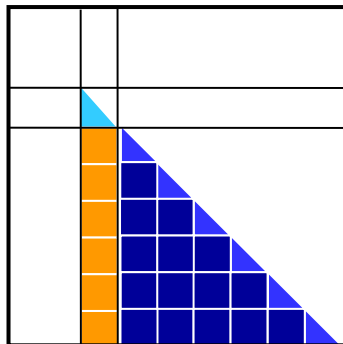


■ Something similar for (dense) linear algebra?



2nd iter.

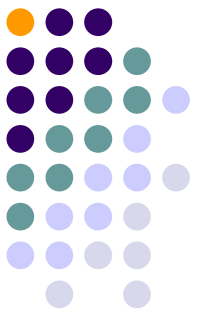
3rd iter.



- Apply “scalar” techniques at the block level
- Software implementation
- Thread/Task-level parallelism
- Target the cores/GPUs of the platform

Leveraging hw. concurrency

Task parallelism

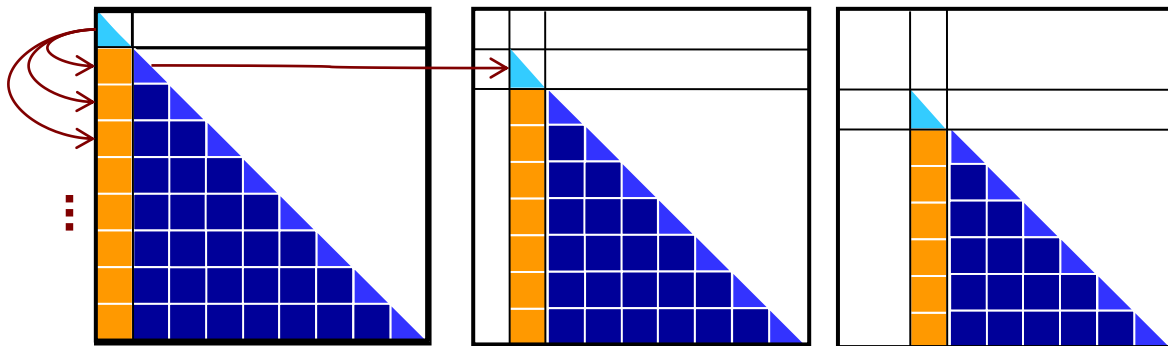


- Read/written blocks determine dependencies, as in scalar case

```
loop:  ld  f0, 0(r1)
      addd f4, f0, f2
      sd  f4, 0(r1)
      addi r1, r1, #8 ...
```

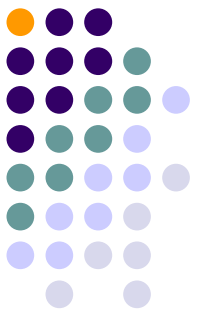
```
for (k=1; k<=n/b; k++){
    Chol A[k,k];
    for (i=k+1; i<=n/b; i++)
        Trsm(A[k,k], A[i,k]); ...
```

Dependencies form a dependency DAG (task tree)



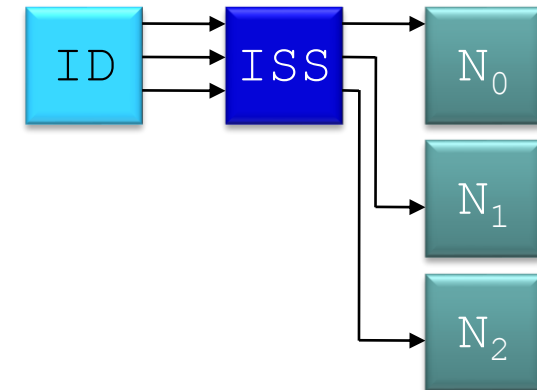
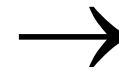
Leveraging hw. concurrency

Task parallelism



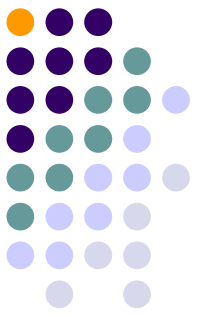
■ *Runtime:*

- Decode (ID): Generate the task tree with a “symbolic analysis” of the code at execution time
- Issue (ISS): Architecture-aware execution of the tasks in the tree

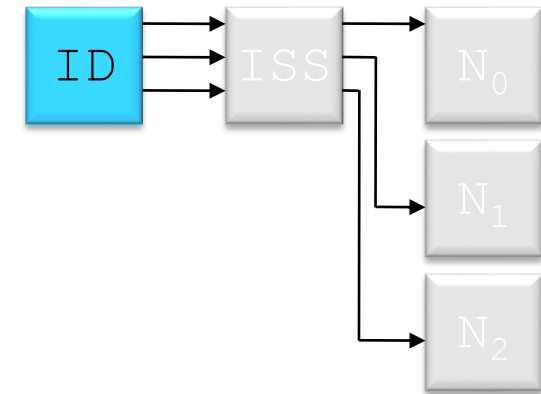


Leveraging hw. concurrency

Task parallelism



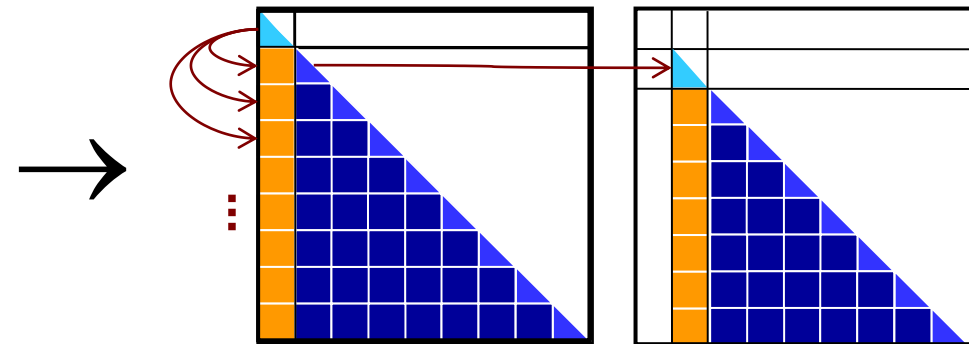
- Decode stage:
 - “Symbolic analysis” of the code



Blocked code:

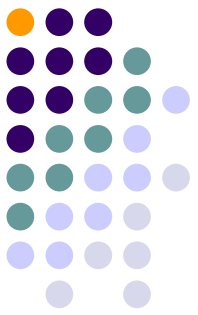
```
for (k=1; k<=n/b; k++) {  
    Chol(A[k,k]);  
    for (i=k+1; i<=n/b; i++)  
        Trsm(A[k,k], A[i,k]); ...  
}
```

Task tree:



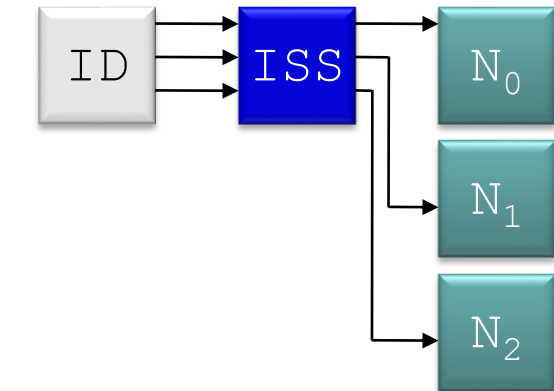
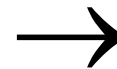
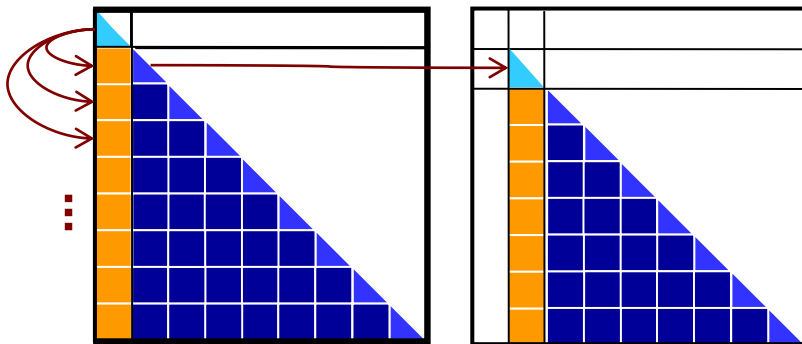
Leveraging hw. concurrency

Task parallelism



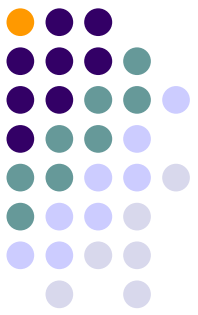
■ Issue stage:

- Temporal scheduling of tasks, attending to dependencies
- Mapping (spatial scheduling) of tasks to resources, aware of locality



Leveraging hw. concurrency

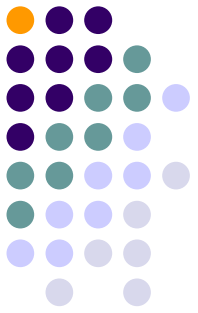
Implementations



- SuperMatrix (UT@Austin and UJI)
 - Read/written blocks defined implicitly by the operations
 - Only valid for dense linear algebra operations encoded in `libflame`
- SMPs (BSC) and GPUSs (BSC and UJI)
 - OpenMP-like languages

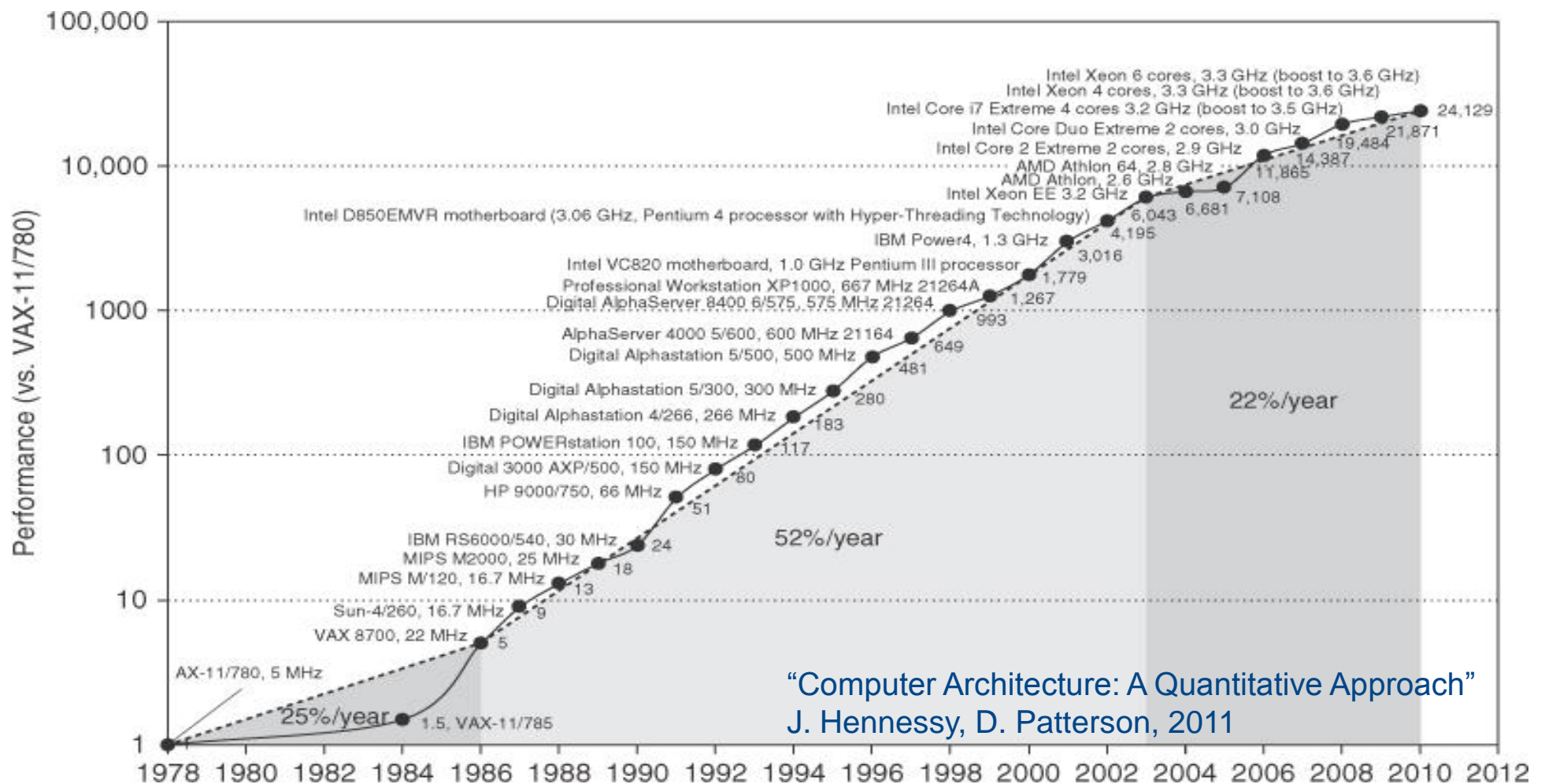
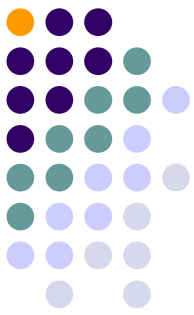
```
#pragma css task inout(A[b*b])  
void Chol(double *A);
```
 - Applicable to task-parallel codes on different platforms: multi-core, multi-GPU, multi-accelerators, Grid,...

Index

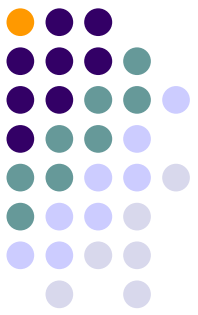


1. Scientific applications
2. Leveraging hardware concurrency
3. **Cost of energy**

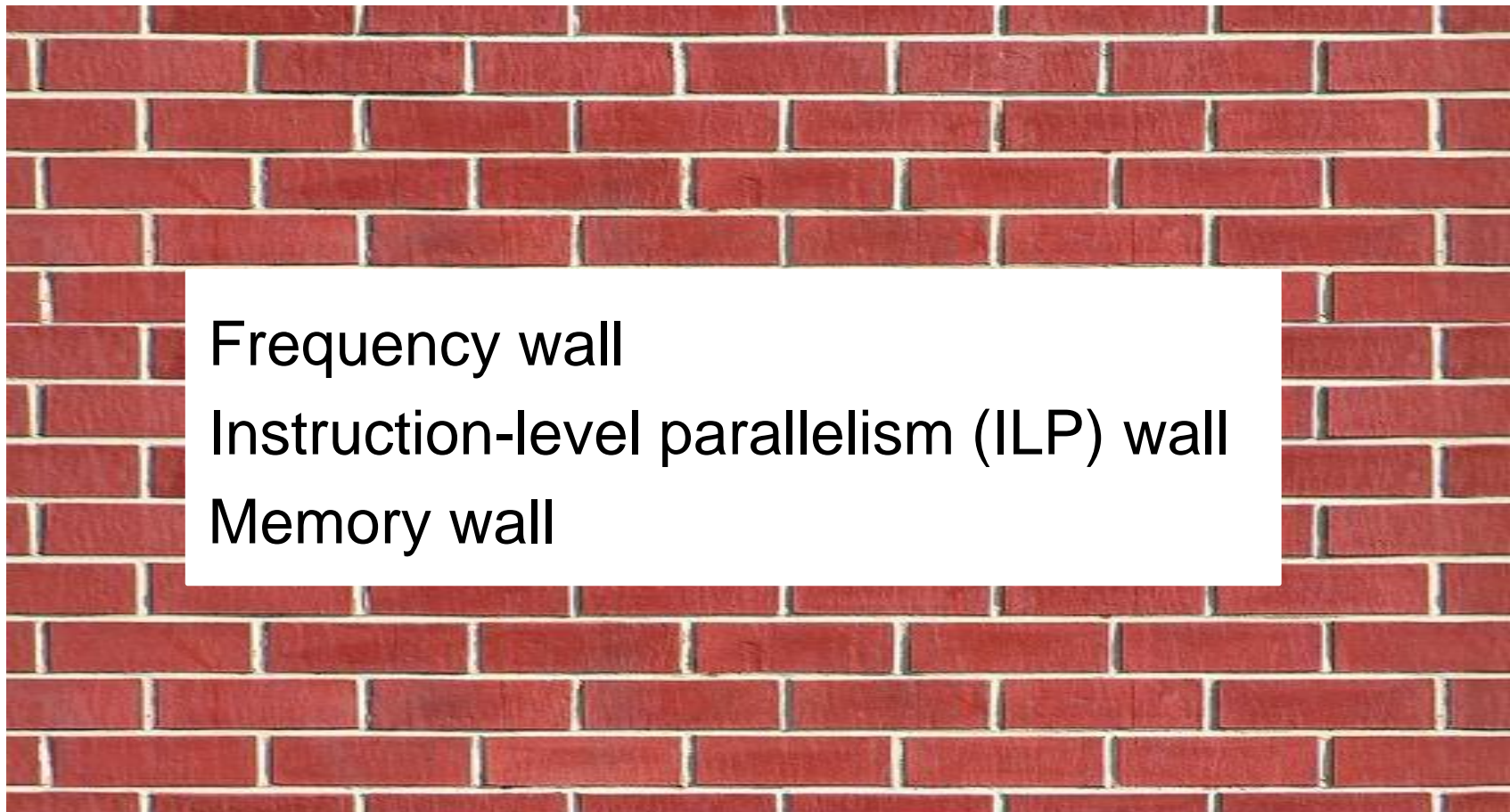
Cost of energy



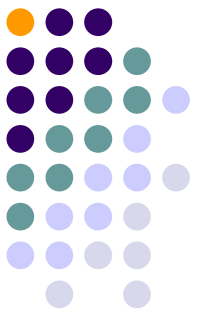
Cost of energy



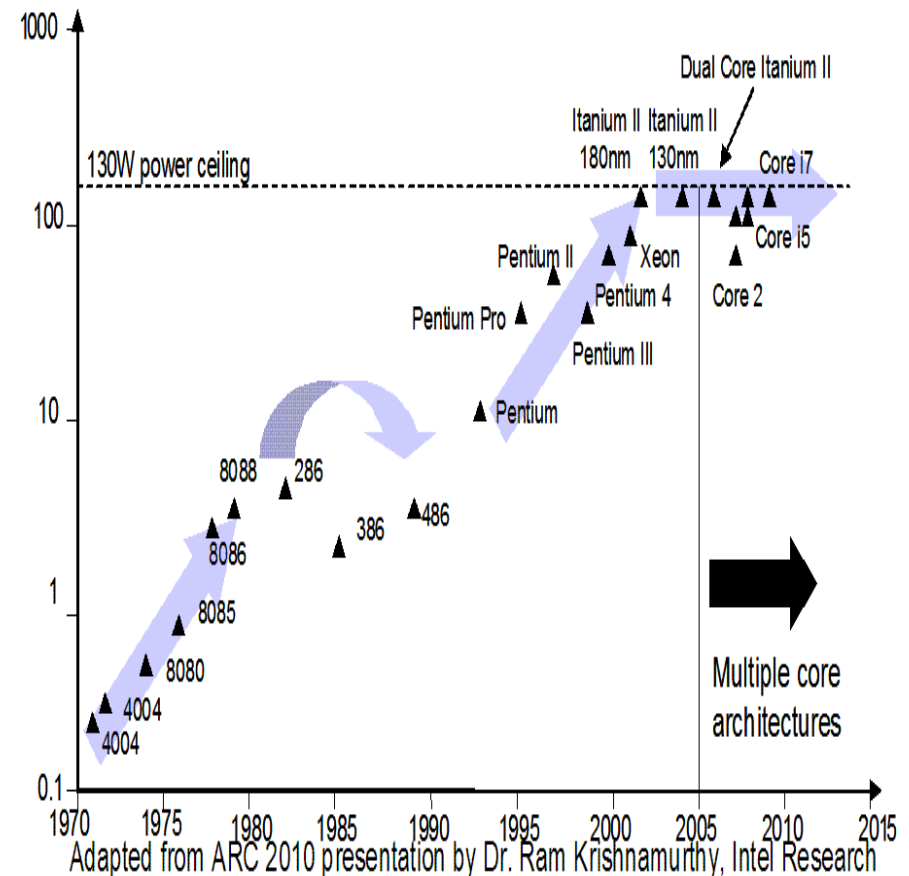
- “The free lunch is over” (H. Sutter, 2005)



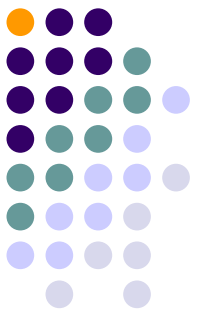
Cost of energy



- Frequency wall
 - Power - energy consumption proportional to $f^3 - f^2$
 - Electricity = money
- 1st Law of Thermodynamics: Energy cannot be created or destroyed, only converted
 - Cost of extracting heat
 - Heat reduces lifetime



Cost of energy



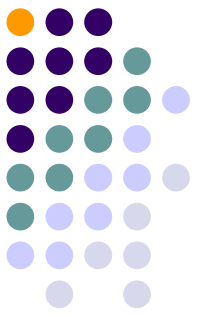
Rank Green/Top	Site	#Cores	MFLOPS/W	LINPACK (TFLOPS)	MW to EXAFLOPS?
1/29	IBM Rochester – BlueGene/Q, Power BQC 16C 1.60 GHz	32.768	2.026.48	339,83	493,47
7/114	BSC (Spain) – Bull B505, Xeon E5649 6C 2.53 GHz, NVIDIA 2090	5.544	1.266,26	103,20	789,73
32/1	RIKEN AICS K Computer– Spar64 Vlllfx (8-core)	705.024	830,18	10.510,00	1.204,60



NVIDIA GTX 480 (250 W) (=1/4 low power hair dryer)
2 million GTXs \approx 493,47 MW!
or 500.000 hair dryers



Cost of energy

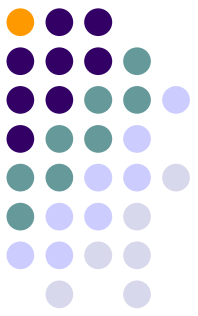


Rank Green/Top	Site	#Cores	MFLOPS/W	LINPACK (TFLOPS)	MW to EXAFLOPS?
1/29	IBM Rochester – BlueGene/Q, Power BQC 16C 1.60 GHz	32.768	2.026,48	339,83	493,47
7/114	BSC (Spain) – Bull B505, Xeon E5649 6C 2.53 GHz, NVIDIA 2090	5.544	1.266,26	103,20	789,73
32/1	RIKEN AICS K Computer– Spar64 Vlllfx (8-core)	705.024	830,18	10.510,00	1.204,60



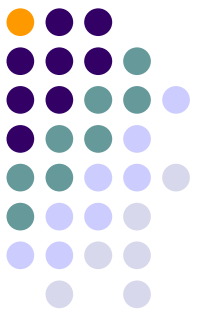
Most powerful reactor under construction in France
Flamanville (EDF, 2017 for US \$9billion):
1,630 MWe

Cost of energy Setup

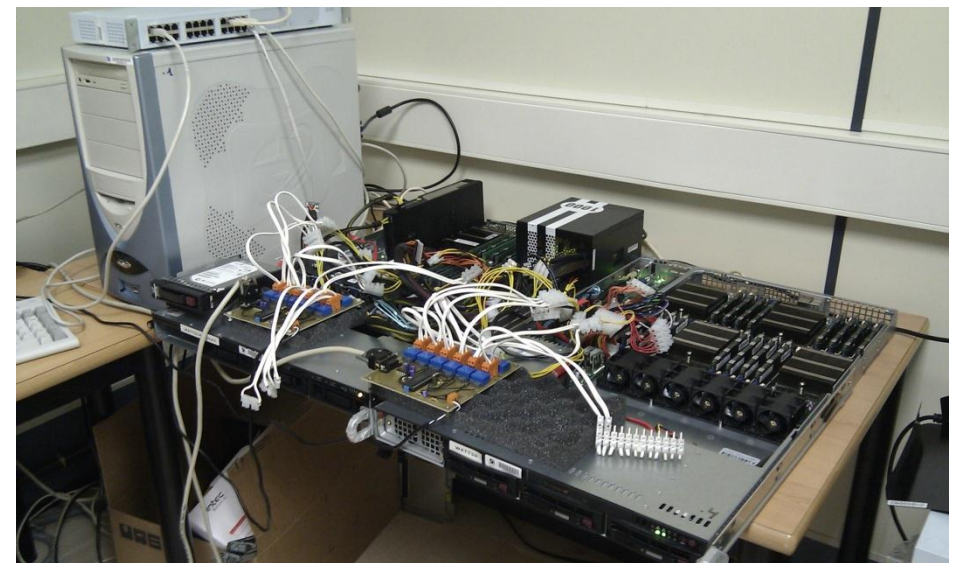
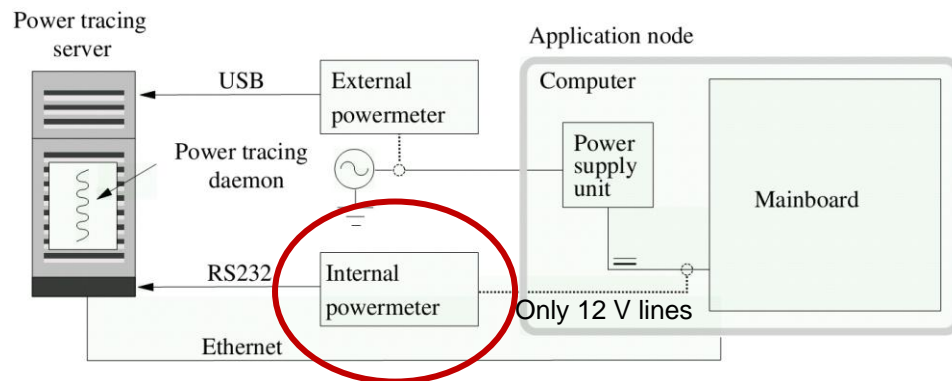


- Modeling power of task-parallel apps.
 - Two Intel Xeon E5504 @ 2.0 GHz (8 cores)
 - Experience: more stable
- Saving opportunities for task-parallel apps.
 - Two AMD Opteron 6128 cores @ 2.0 GHz (16 cores)
 - Experience: more flexible (DVFS at core level)

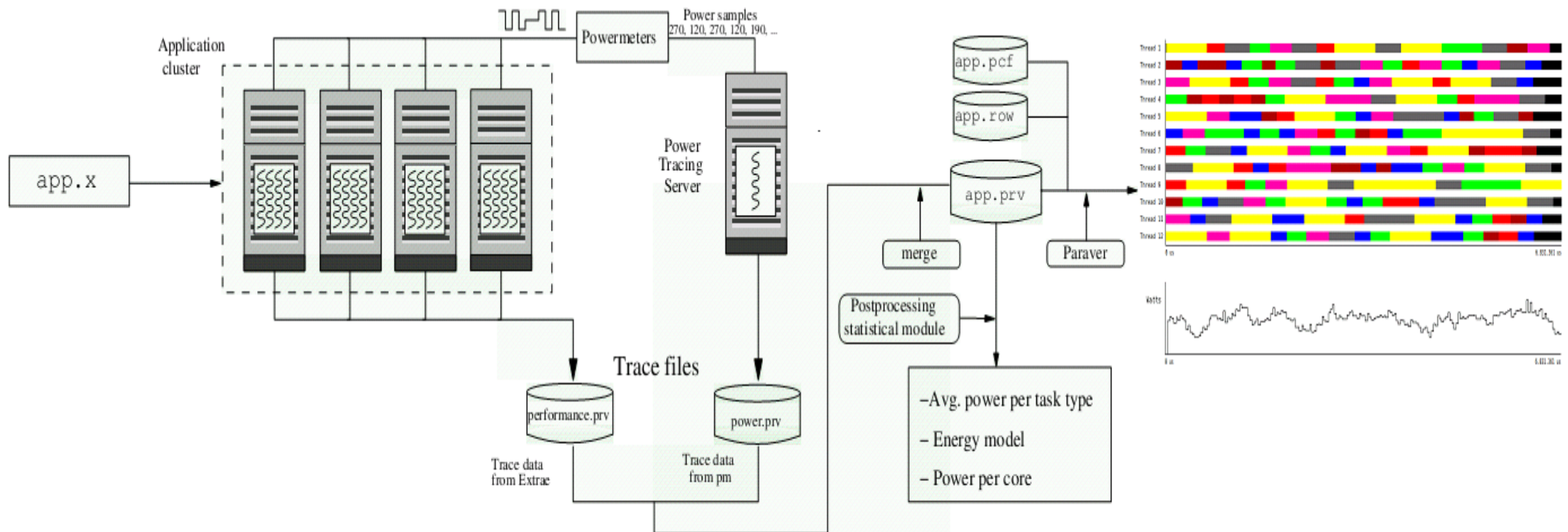
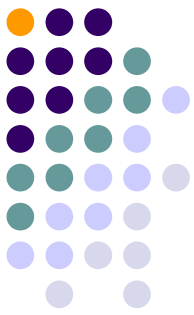
Cost of energy Setup



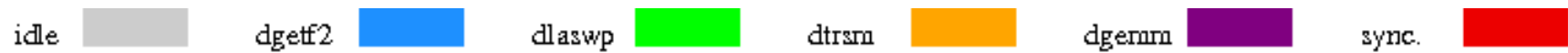
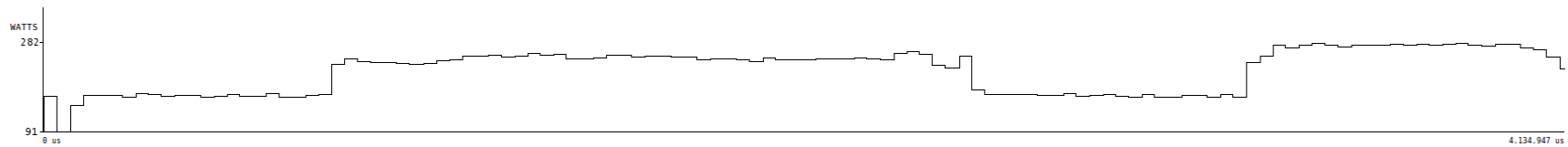
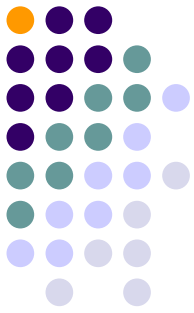
- DC powermeter with sampling freq. = 25 Hz
 - LEM HXS 20-NP transducers with PIC microcontroller
 - RS232 serial port



Cost of energy Setup

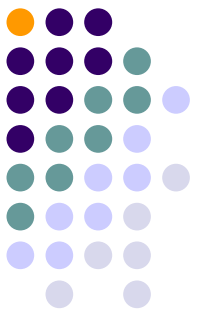


Cost of energy Setup

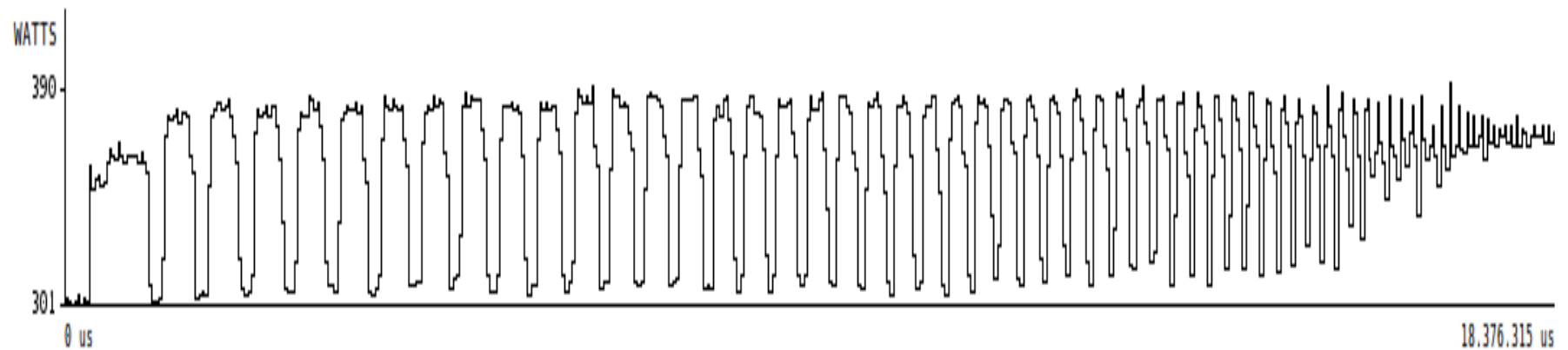


Cost of energy

Power vs. energy

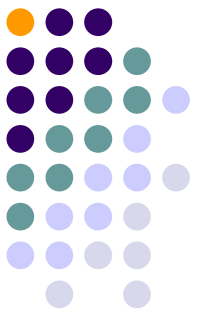


$$E = \int_0^T P dt$$

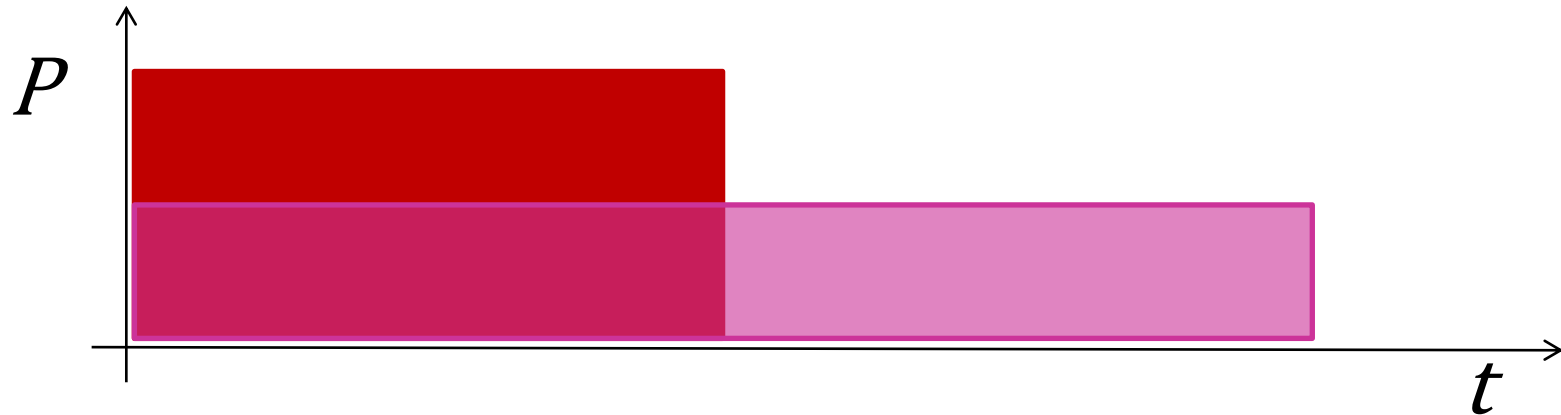


Cost of energy

Power vs. energy

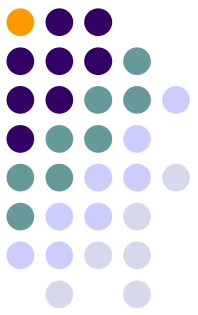


- Which one is better, *A* or *B*?



$$E_A = P_A t_A$$

$$E_B = P_B t_B$$



Cost of energy

Modeling power (mainboard)

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

p^C is power dissipated by CPU (socket): $p^S + p^D$

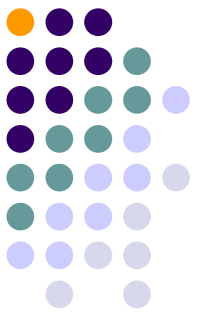
p^Y is power of remaining components (e.g., RAM)

Considerations:

- p^Y and p^S are constants (though p^S grows with temperature)
- Hot system
- Task-parallel routines
- Intel platform

Cost of energy

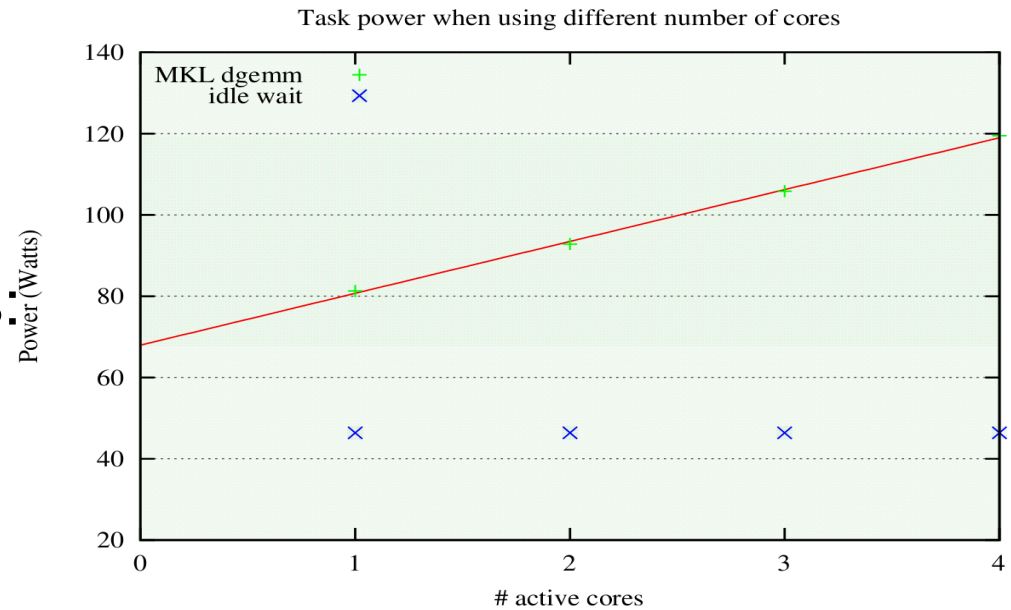
Modeling power (mainboard)



- System power:

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

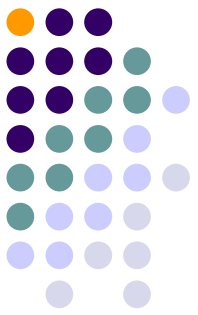
Estimated as *idle* power
Due to off-chip components:
e.g, RAM (only mainboard)



$$P^Y \approx P^I = 46.37 \text{ W}$$

Cost of energy

Modeling power (mainboard)

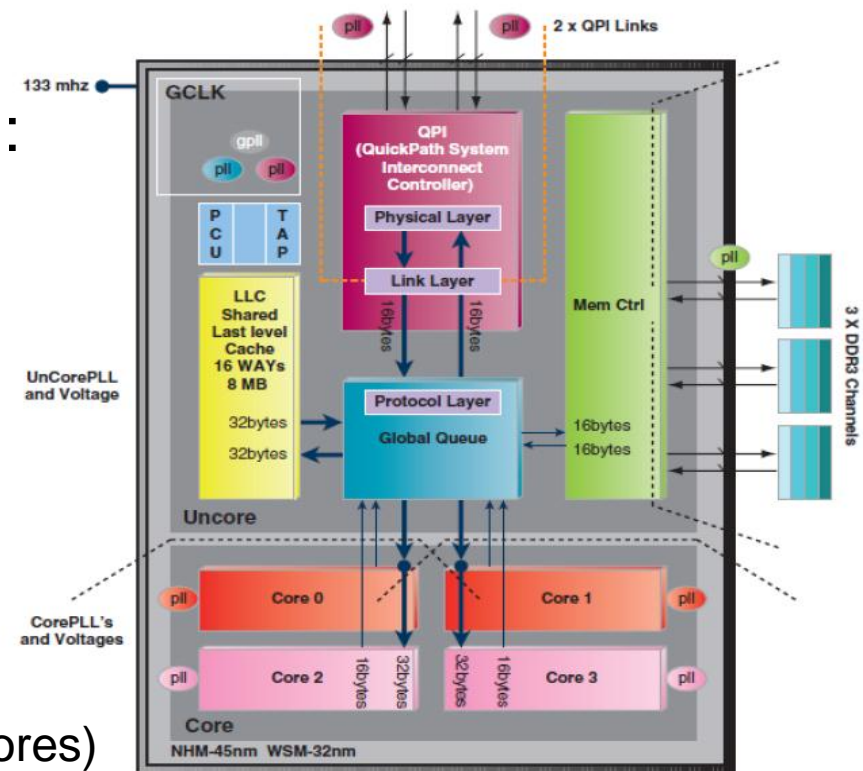


- Static power:

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

Also known as *Uncore* power (Intel):

- LLC
- Mem. controller
- Interconnect controller
- Power control logic
- etc.

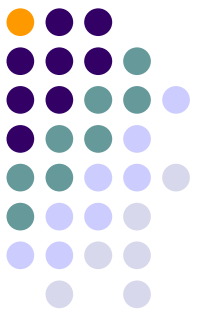


Intel Xeon 5500 (4 cores)

The Uncore: A Modular Approach to Feeding the High-performance Cores.
D. L. Hill et al. Intel Technology Journal, Vol. 14(3), 2010

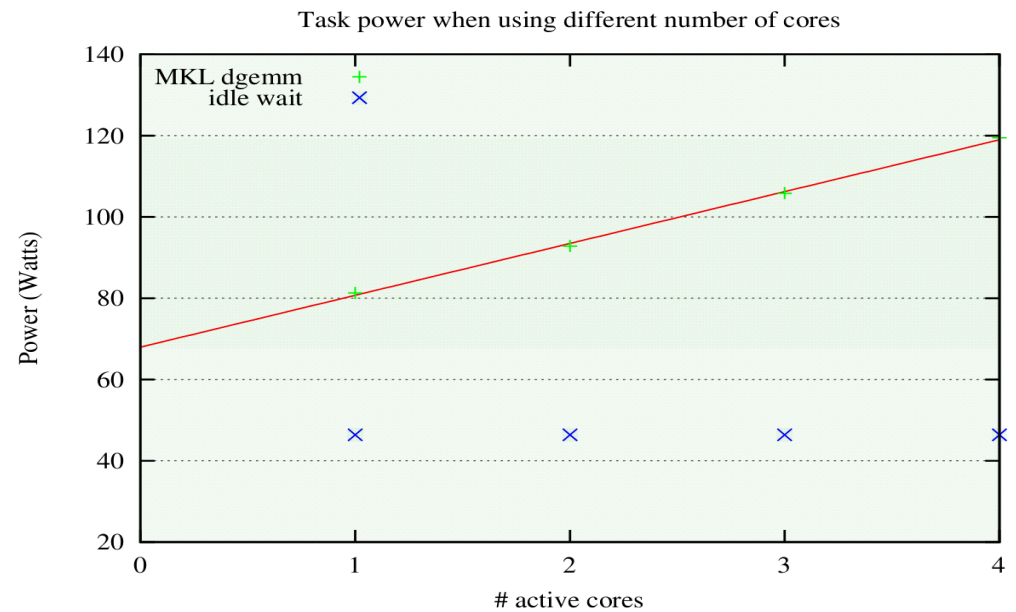
Cost of energy

Modeling power (mainboard)



- Static power:

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

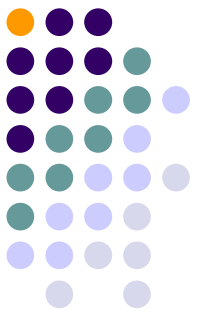


$$P_{dgemm}(c) = 67.97 + 12.75 c$$

$$P^S = 67.97 - 46.37 = 21.6 \text{ W}$$

Cost of energy

Modeling power (mainboard)

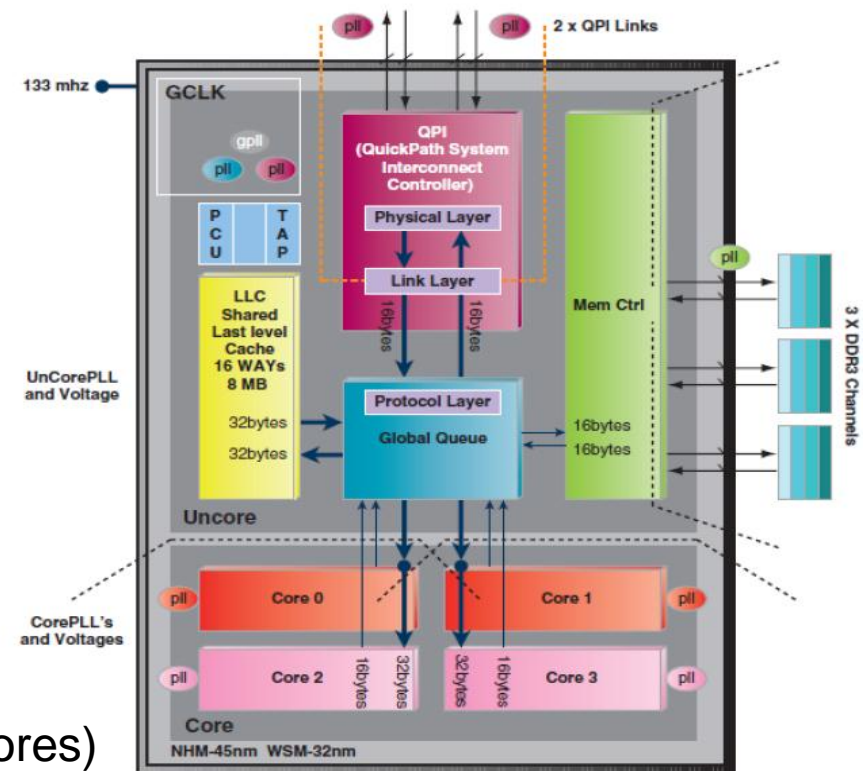


- Dynamic power:

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

Also known as *Core power* (Intel):

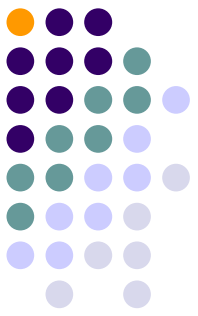
- Execution units
- L1 and L2 cache
- Branch prediction logic
- etc.



Intel Xeon 5500 (4 cores)

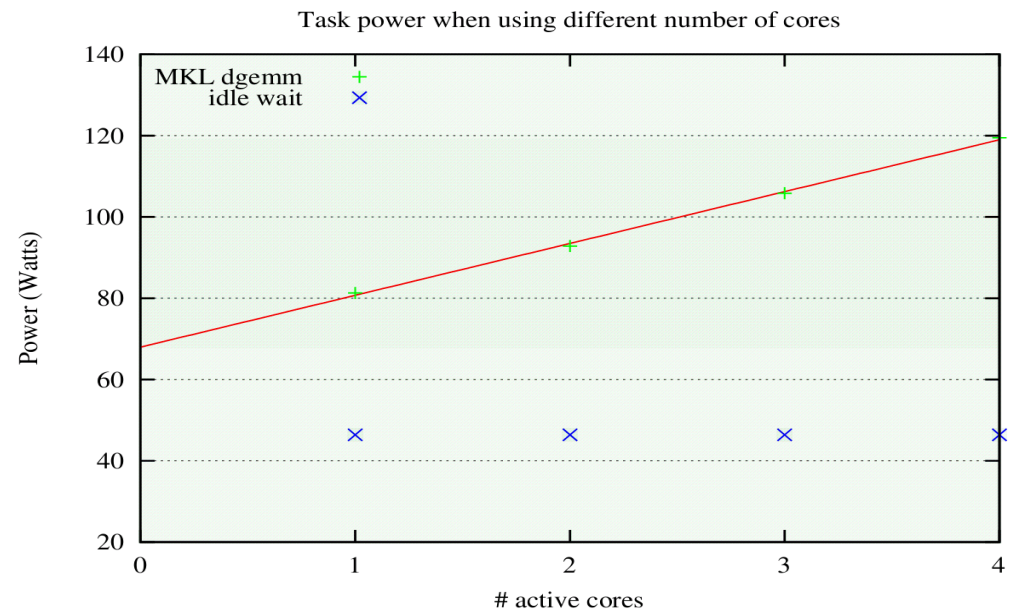
Cost of energy

Modeling power (mainboard)



- Dynamic power:

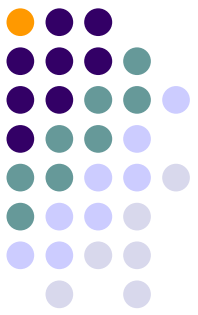
$$P = P^Y + P^S + P^D$$



$$P_{dgemm}(c) = 67.97 + 12.75 c$$

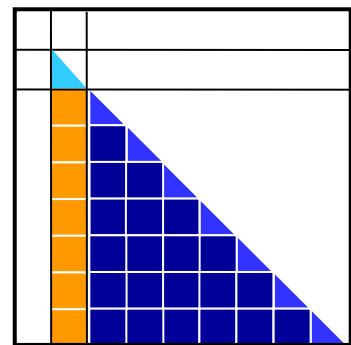
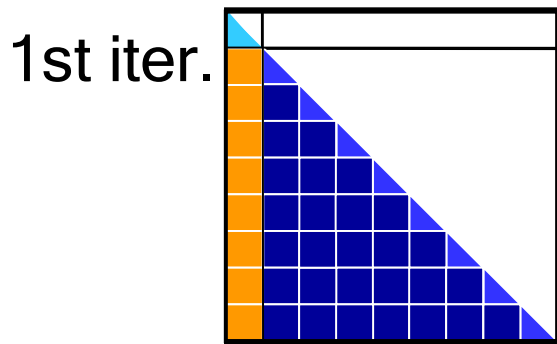
Cost of energy

Modeling power (mainboard)



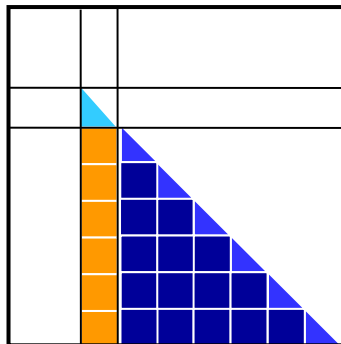
- Dynamic power of task-parallelCholesky

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



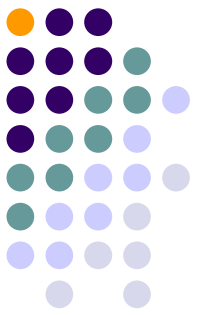
2nd iter.

3rd iter.



```

for (k=1; k<=n/b; k++){
  F:  Chol(A[k,k]);
      for (i=k+1; i<=n/b; i++){
        T:  Trsm(A[k,k], A[i,k]);
          for (i=k+1; i<nb; i++){
            U1:  Syrk(A[i,k], A[i,i]);
              for (j=k+1; j<=i; j++){
                U2:  Gemm(A[i,k], A[j,k], A[i,j]);
              }
            }
          }
        }
      }
    }
  
```

Cost of energy

Modeling power (mainboard)

- Dynamic power of task-parallelCholesky

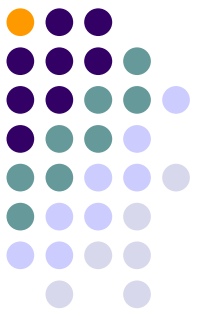
For a given kernel, execute repeatedly till power stabilizes:

$$P^D_{dgemm} = P_{dgemm} - (P^Y + P^S)$$

Task	1 kernel mapped to 1 core				2 kernels mapped to 2 cores of different sockets			
	Block size, b				Block size, b			
	128	192	256	512	128	192	256	512
P^D_P (dpotrf)	10.26	10.35	10.45	11.28	9.05	9.09	9.28	10.44
P^D_T (dtrsm)	10.12	10.31	10.32	10.80	9.45	9.57	9.60	11.08
P^D_S (dsyrk)	11.22	11.47	11.67	12.60	10.42	10.63	10.82	11.80
P^D_G (dgemm)	11.98	12.54	12.72	13.30	10.90	12.16	11.28	11.96
P^D_B (busy)	7.62	7.62	7.62	7.62	7.62	7.62	7.62	7.62

Power increases linearly with #cores, from 1 to 4 mapped to a single socket

When two sockets are used, linear function changes



Cost of energy

Modeling power (mainboard)

- Power of task-parallel Cholesky

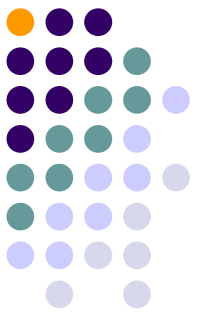
$$P_{chol} = P^Y + P^S + \sum_{i=1}^r \sum_{j=1}^c P^D_i N_{i,j}(t)$$

where

- r is #different types of tasks
- c is #cores
- P^D_i is the average dynamic power for task of type i
- $N_{i,j}(t) = 1$ if thread j is executing a task of type i at time t ; $N_{i,j}(t) = 0$ otherwise

Cost of energy

Modeling power (mainboard)



- Energy of task-parallel Cholesky

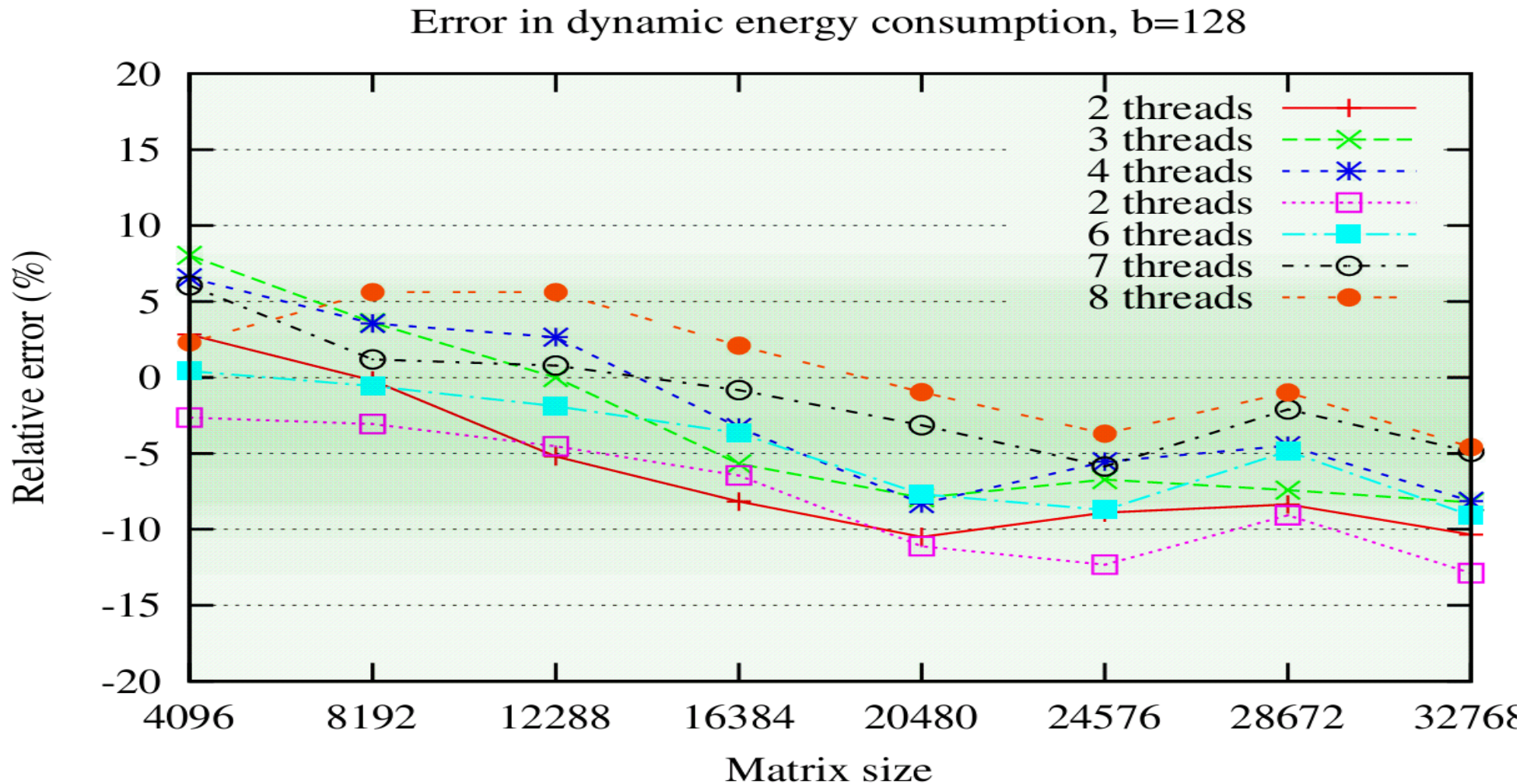
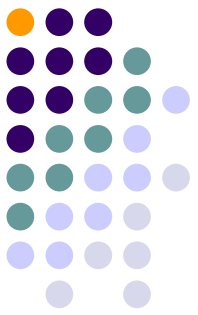
$$\begin{aligned} E_{chol} &= (P^Y + P^S) T + \sum_{i=1}^r \sum_{j=1}^c P^D_i \int_{t=0}^T N_{i,j}(t) dt \\ &= (P^Y + P^S) T + \sum_{i=1}^r \sum_{j=1}^c P^D_i T_{i,j} \end{aligned}$$

where

- T is the total execution time
- $T_{i,j}(t) = 1$ is the time that thread j has executed tasks of type i

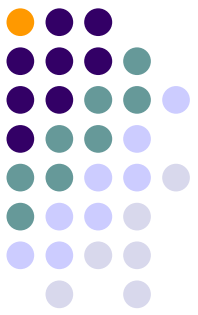
Cost of energy

Modeling power (mainboard)

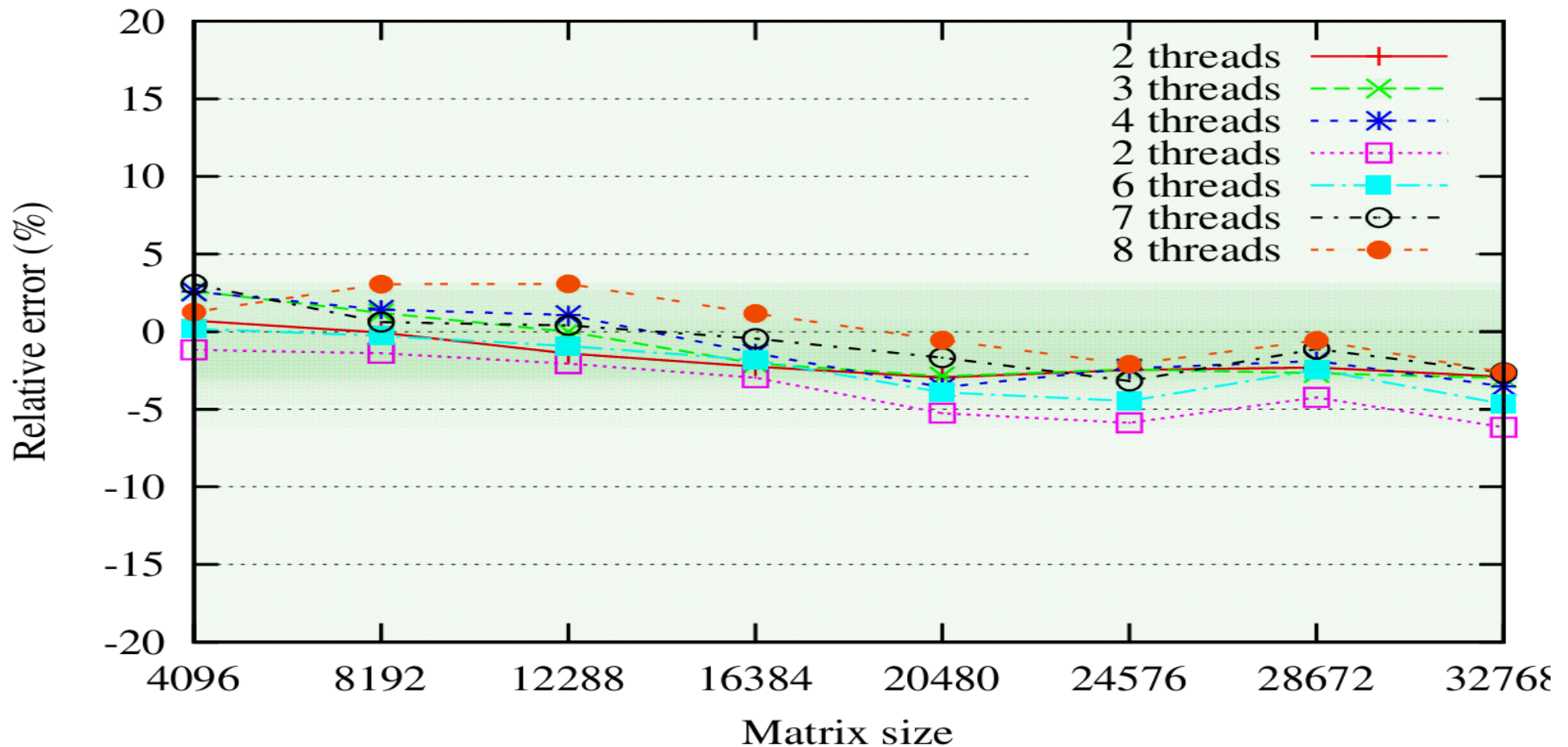


Cost of energy

Modeling power (mainboard)

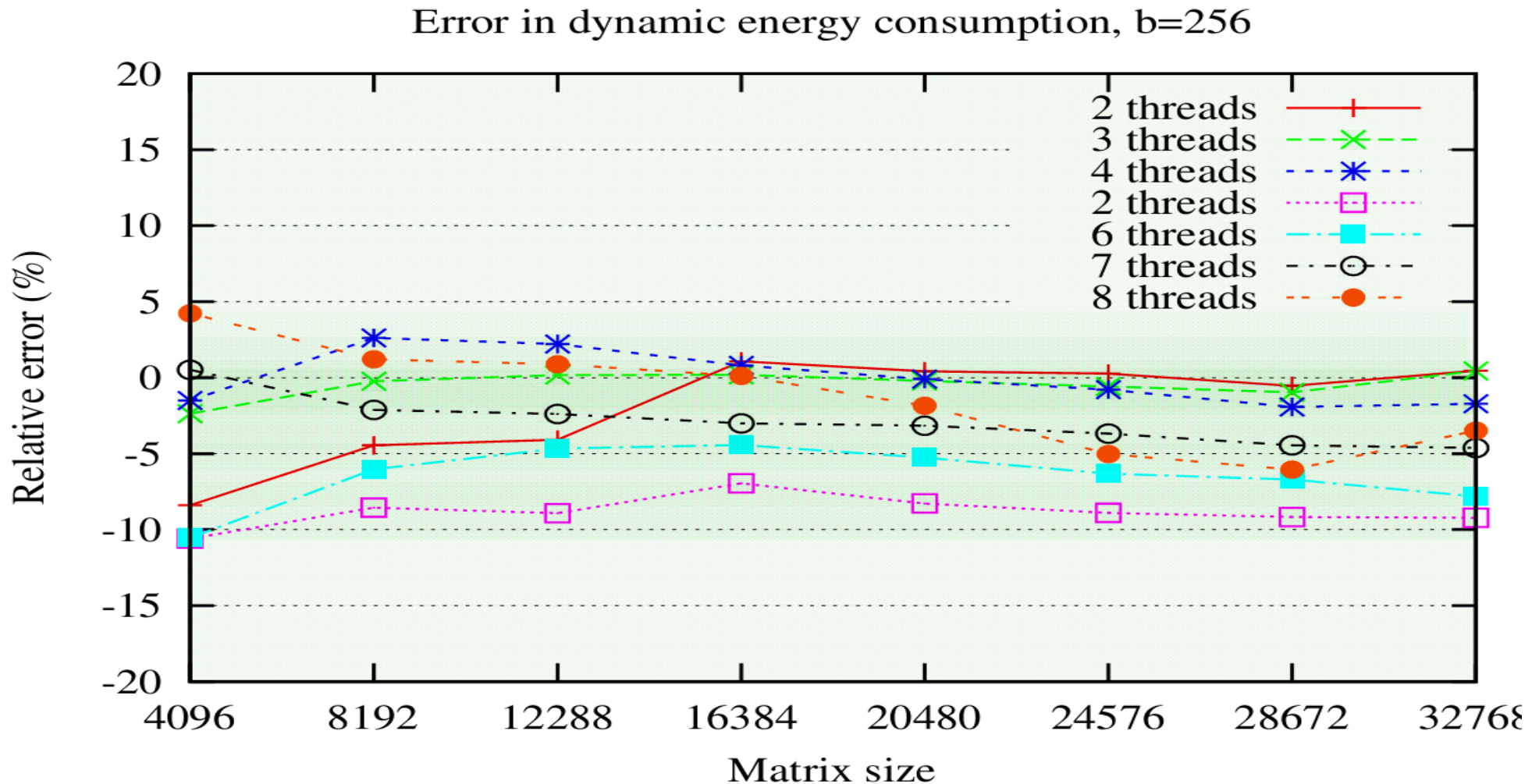
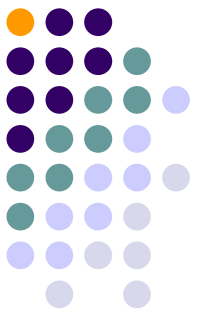


Error in total energy consumption, $b=128$



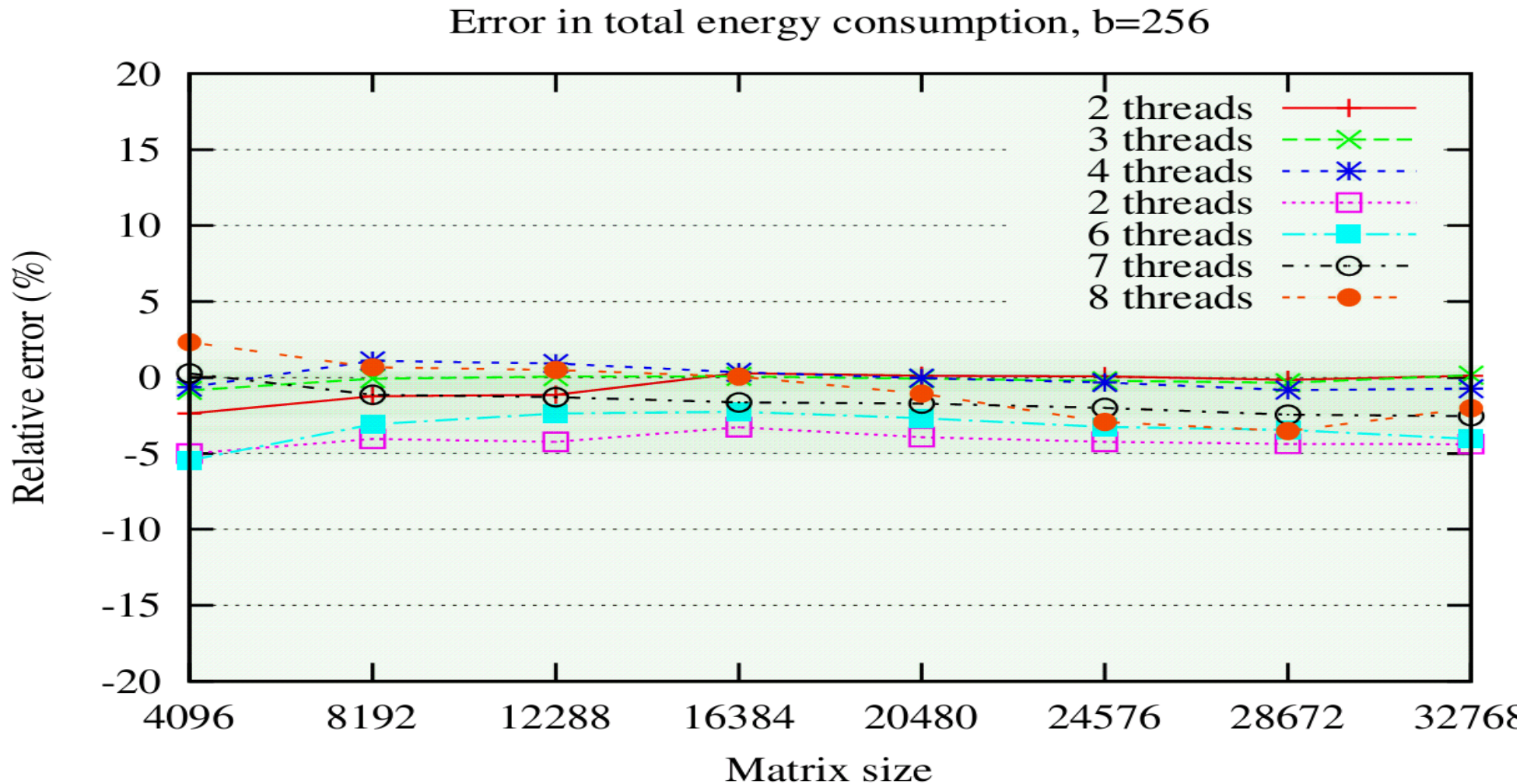
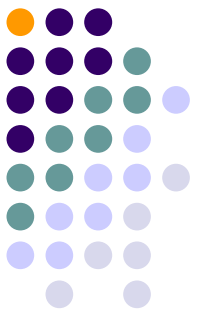
Cost of energy

Modeling power (mainboard)



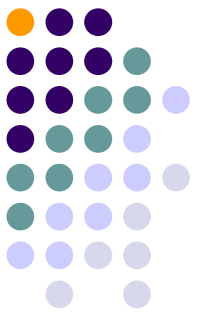
Cost of energy

Modeling power (mainboard)



Cost of energy

Saving opportunities



- ACPI (*Advanced Configuration and Power Interface*): industry-standard interfaces enabling OS-directed configuration, power/thermal management of mobile/desktop/server platforms



Microsoft

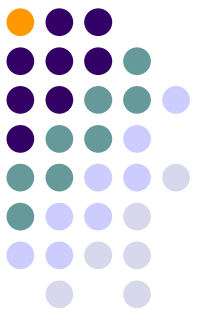


TOSHIBA

- Revision 5.0 (december 2011)
- In the processor: Power states (C-states) and performance states (P-states)

Cost of energy

Saving opportunities



- Power states (C-states):
 - C0: normal execution (also a P-state)
 - Cx, $x > 0$: no instructions being executed. As x grows, more savings but longer latency to reach C0
 - Stop clock signal
 - Flush and shutdown cache (L1 and L2 flushed to LLC)
 - Turn off cores

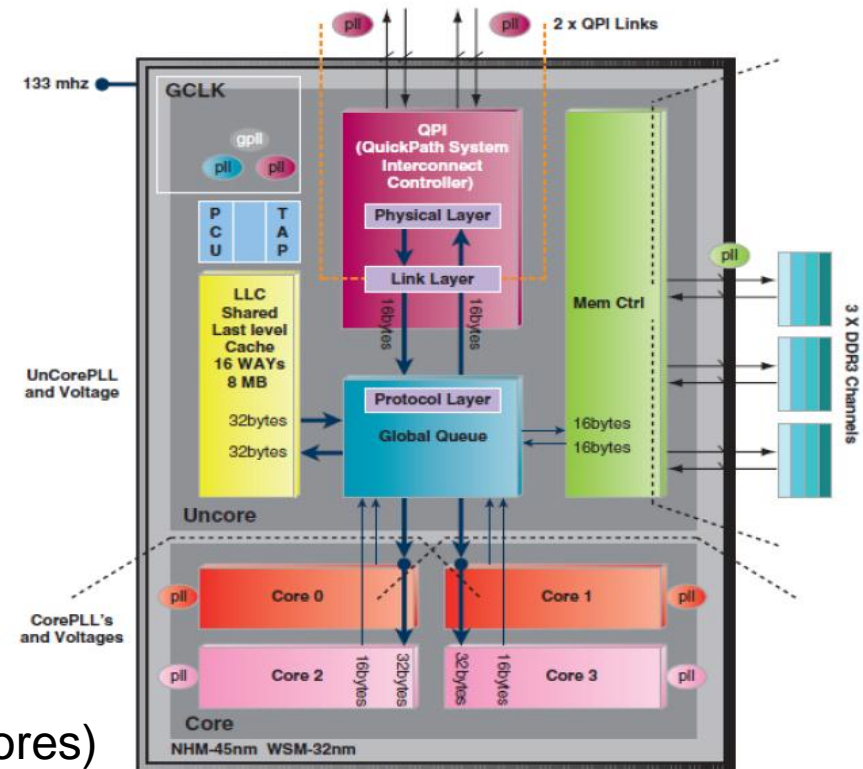
Cost of energy

Saving opportunities

- Package power states (PC-states):
 - PC0, PC1, PC2,...

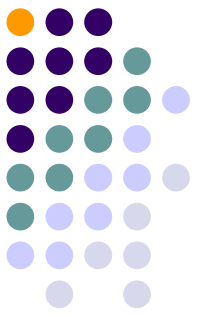
Uncore subsystem remains active and consumes power as long as there is any active core on the CPU

Intel Xeon 5500 (4 cores)



Cost of energy

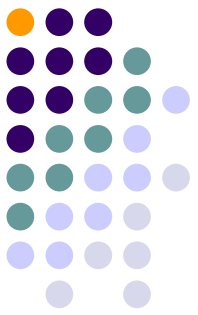
Saving opportunities



- Intel Core i7 processor:
 - **Core C0 State**
 - The normal operating state of a core where code is being executed
 - **Core C1/C1E State**
 - The core halts; it processes cache coherence snoops
 - **Core C3 State**
 - The core flushes the contents of its L1 instruction cache, L1 data cache, and L2 cache to the shared L3 cache, while maintaining its architectural state. All core clocks are stopped at this point. No snoops
 - **Core C6 State**
 - Before entering core C6, the core will save its architectural state to a dedicated SRAM on chip. Once complete, a core will have its voltage reduced to zero volts

Cost of energy

Saving opportunities



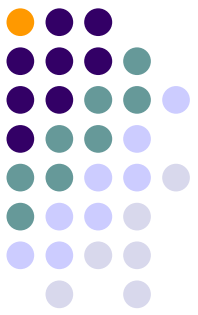
- Performance states (P-states):
 - P0: Highest performance and power
 - $P_i, i > 0$: As i grows, more savings but lower performance

P-state P_i	VCC_i	f_i	AMD platform
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	

- $P = a V^2 f$, where a depends on the technology (but
 $E = \int_0^T P dt = a V^2$) \longrightarrow DVFS!

Cost of energy

Saving opportunities

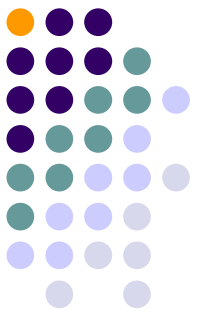


- Leveraging DVFS: cpufreq

```
quintana@watts2:~$ cpufreq-info
analyzing CPU 15:
  driver: powernow-k8
  CPUs which run at the same hardware frequency: 15
  CPUs which need to have their frequency coordinated by software: 15
  maximum transition latency: 10.0 us.
  hardware limits: 800 MHz - 2.00 GHz
  available frequency steps: 2.00 GHz, 1.50 GHz, 1.20 GHz, 1000 MHz, 800 MHz
  available cpufreq governors: ondemand, userspace, performance
  current policy: frequency should be within 800 MHz and 2.00 GHz.
                   The governor "ondemand" may decide which speed to use
                   within this range.
  current CPU frequency is 800 MHz (asserted by call to hardware).
```

Cost of energy

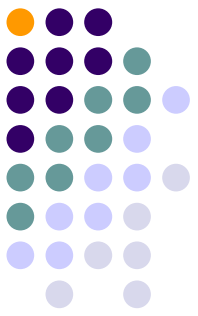
Saving opportunities



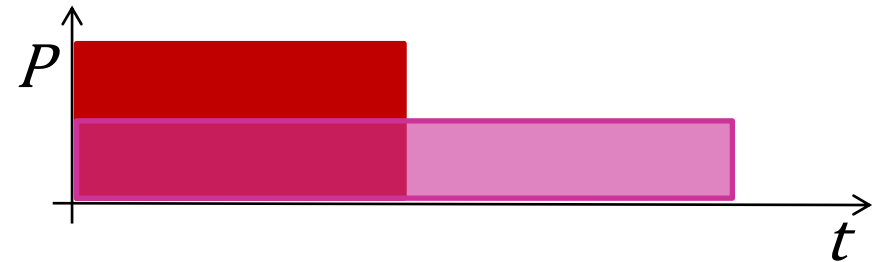
- Leveraging DVFS (transparent): Linux governors
 - **Performance:** Highest frequency/performance
 - **Powersave:** Lowest frequency/performance
 - **Userspace:** User's decision
 - **Ondemand:** If CPU utilization rises above the threshold value set in the `up_threshold` parameter, the `ondemand` governor increases the CPU frequency to `scaling_max_freq`. When CPU utilization falls below this threshold, the governor decreases the frequency in steps. Lowest performance, growing with workload
 - **Conservative:** If CPU utilization is above `up_threshold`, this governor will step up the frequency to the next highest frequency below or equal to `scaling_max_freq`. If CPU utilization is below `down_threshold`, this governor will step down the frequency to the next lowest frequency until it reaches `scaling_min_freq`

Cost of energy

Saving opportunities

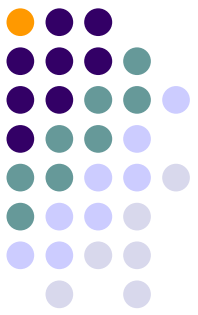


- Which one is better, *A* or *B*?

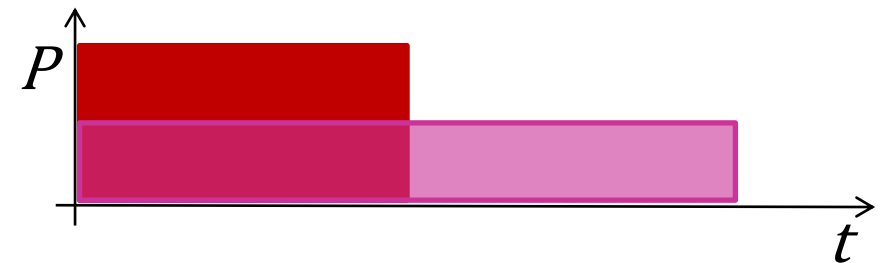
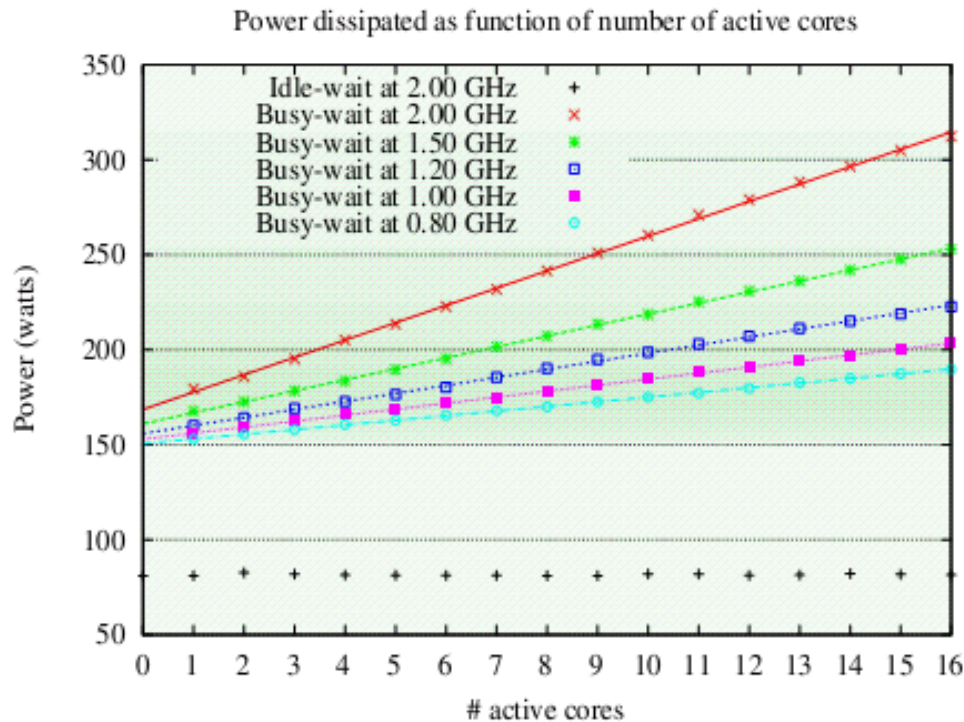


Cost of energy

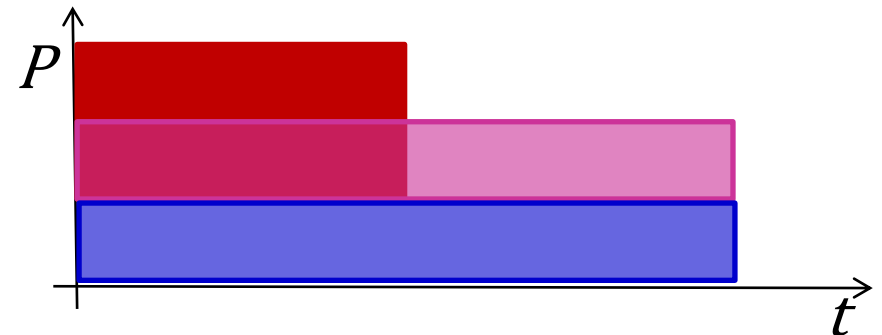
Saving opportunities



- Which one is better, *A* or *B*?

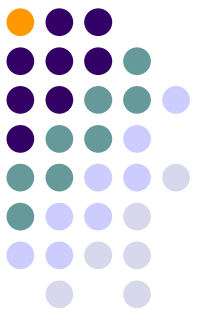


But consider also
 $p^Y + p^S \simeq 50\%$ of power

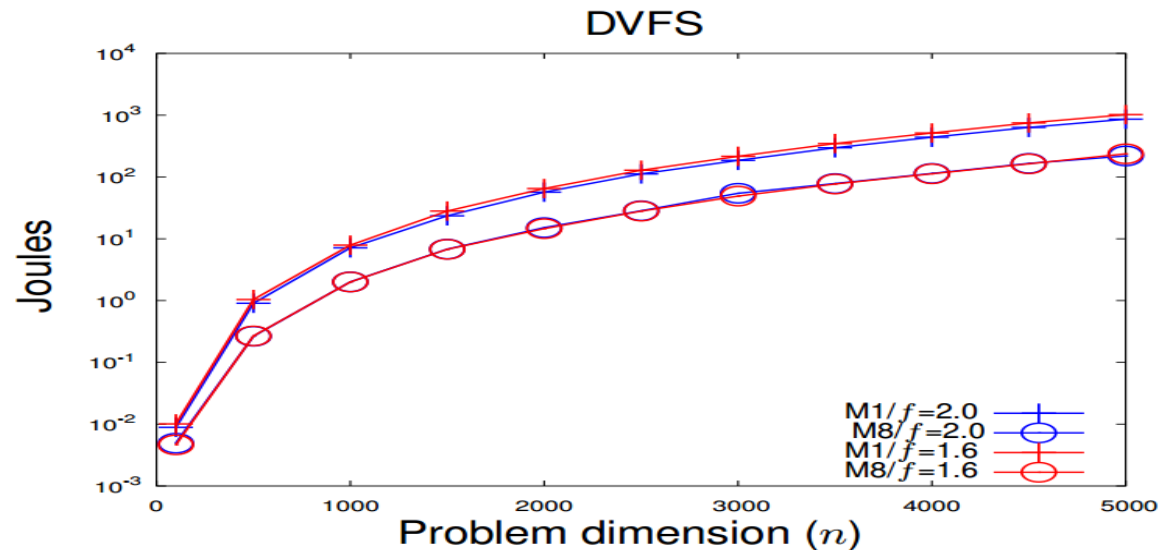


Cost of energy

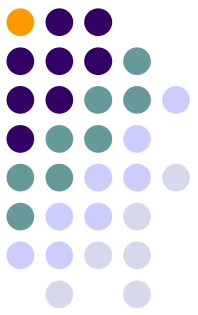
Saving opportunities



- To DVFS or not? General consensus:
 - No for compute-intensive apps.: reducing frequency increases execution time linearly



- Yes for memory-bounded apps. as cores are idle most of the time



Cost of energy

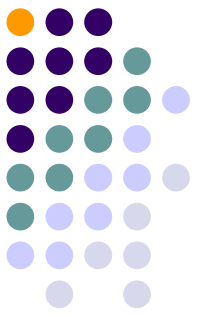
Saving opportunities

- ...but, in many platforms, reducing frequency via DVFS also reduces memory bandwidth proportionally!

P-state P_i	VCC_i	f_i	α_i	β_i	ΔP_i^S	ΔP_i^D	$\Delta P_i^T(16)$	BW_i	ΔBW_i
P_0	1.23	2.00	168.59	9.12	—	—	—	4.43	—
P_1	1.17	1.50	161.10	5.77	-9.52	-32.14	-17.58	3.89	-12.19
P_2	1.12	1.20	155.90	4.23	-17.09	-50.25	-28.34	3.49	-21.21
P_3	1.09	1.00	152.94	3.15	-21.47	-60.73	-33.26	3.19	-27.99
P_4	1.06	0.80	150.61	2.44	-25.73	-70.30	-39.85	2.80	-36.79

Cost of energy

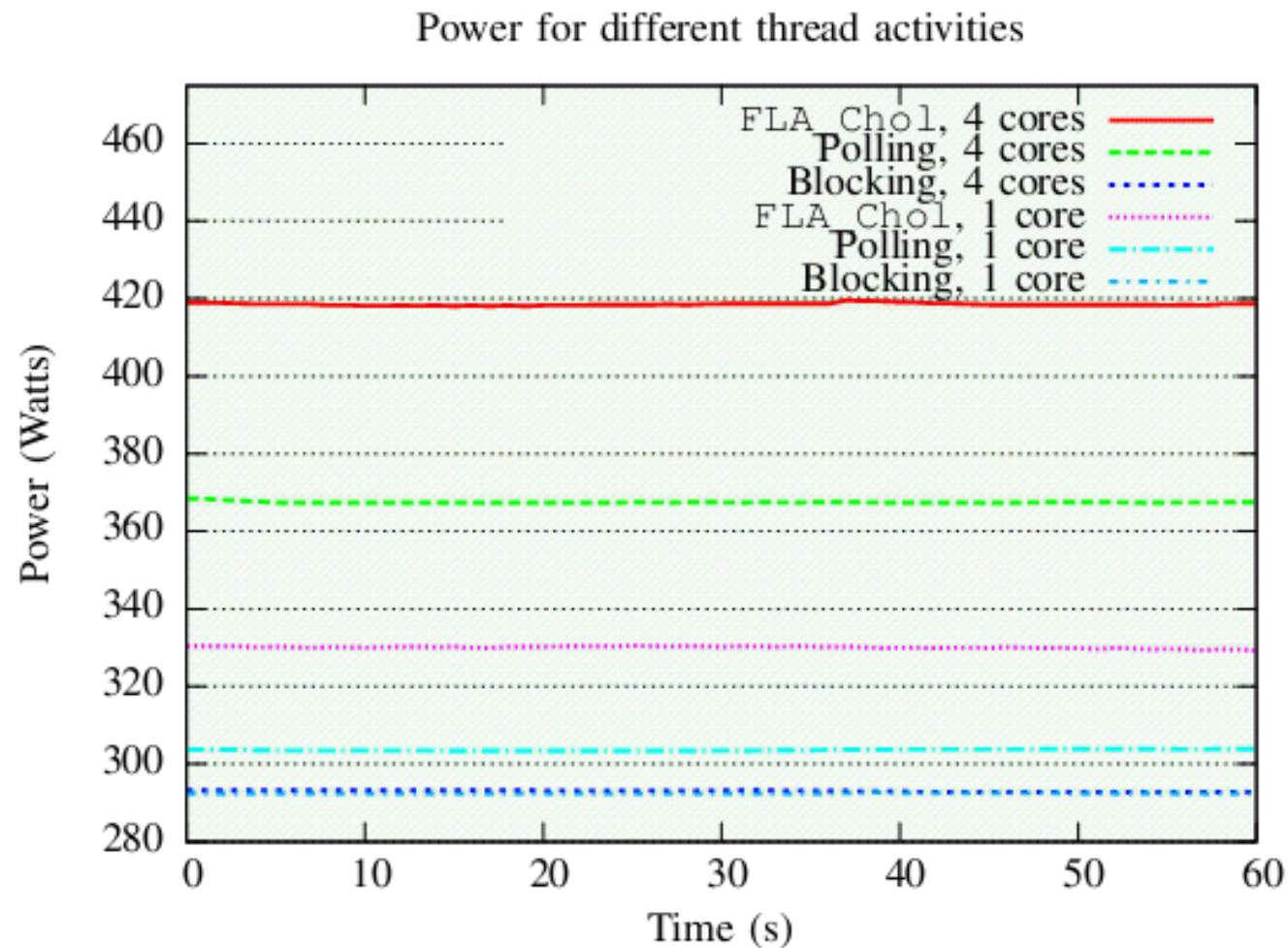
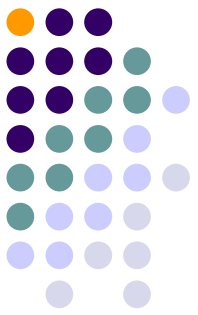
Saving opportunities



- Alternative strategies for compute-intensive apps.:
 - Idle-wait in multithreaded apps.
 - Idle-wait in hybrid CPU-GPU apps.
 - Idle-wait during communications in MPI apps.

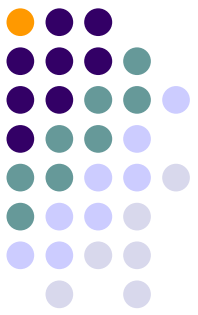
Cost of energy

Saving opportunities

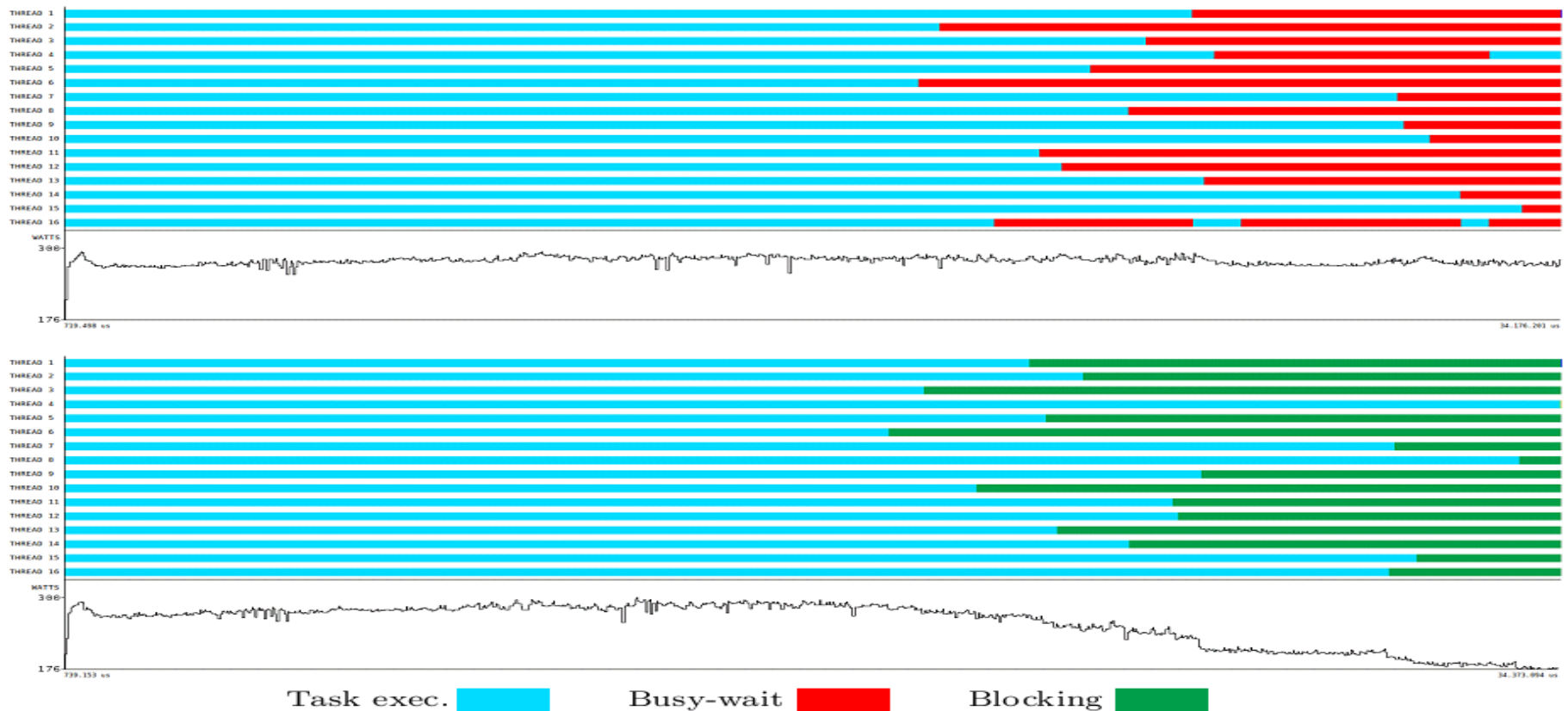


Cost of energy

Saving opportunities

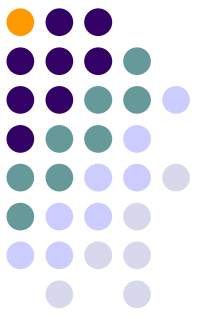


- Idle-wait in multithreaded apps. (ILU preconditioner)

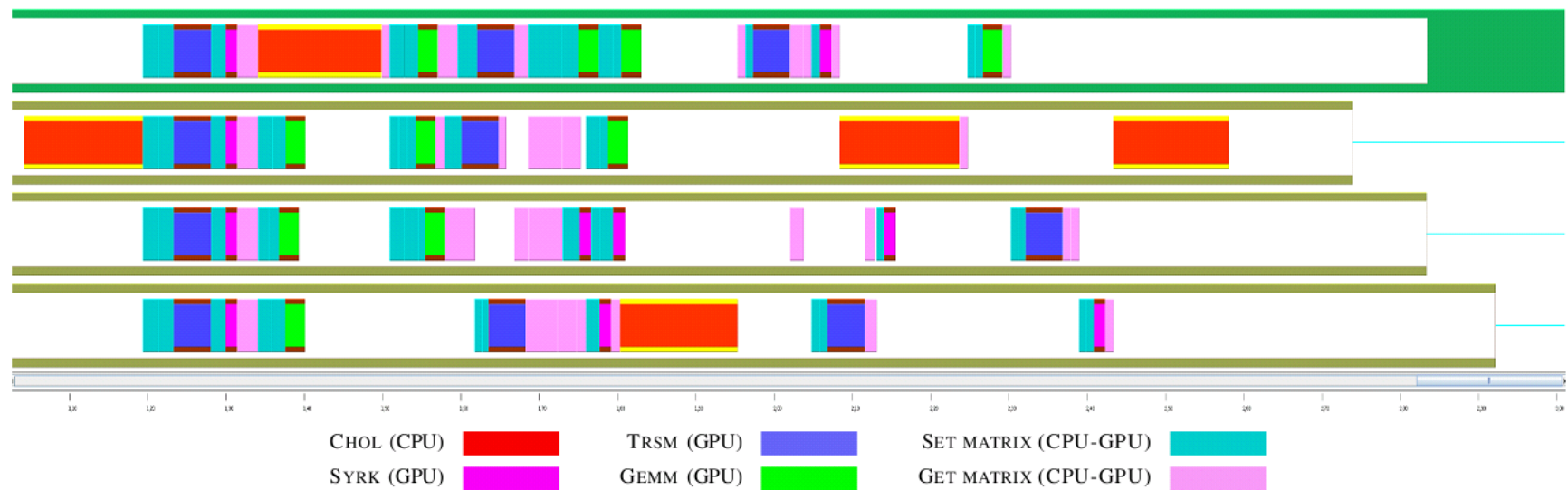


Cost of energy

Saving opportunities



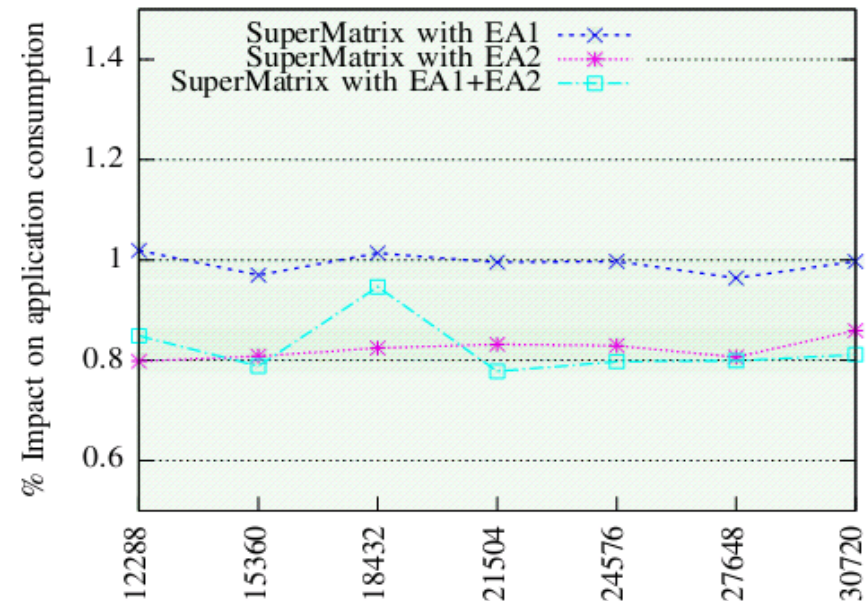
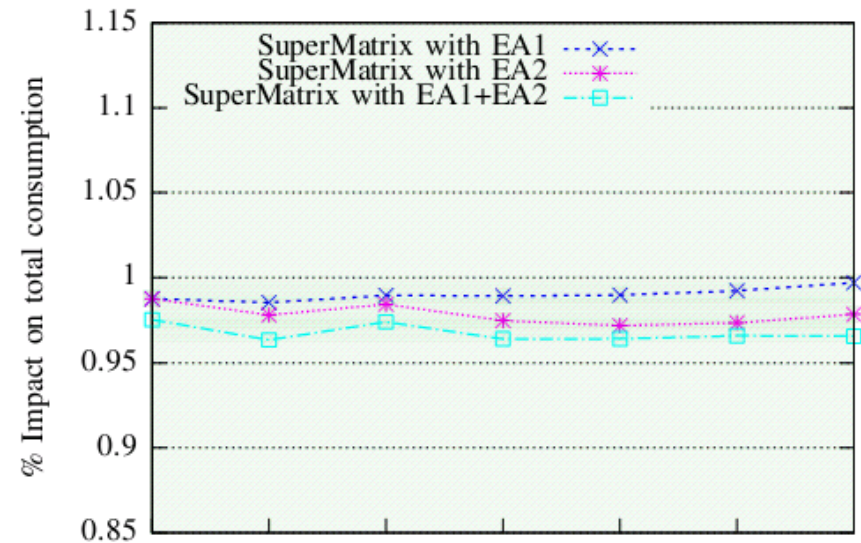
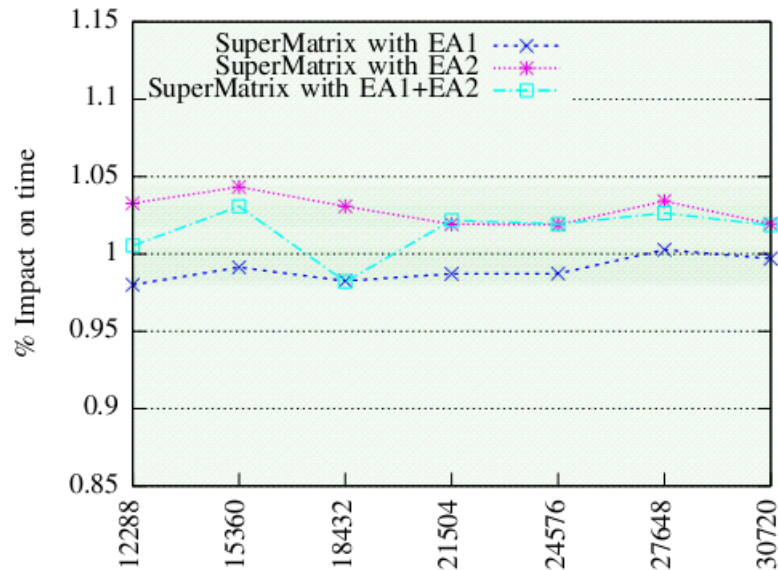
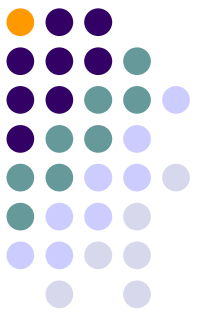
- Idle-wait in hybrid CPU-GPU apps. (multi-GPU Cholesky factorization via SuperMatrix runtime)



- Intel Xeon E5540 @ 2.83 GHz (4 cores) and NVIDIA Tesla S2050 (4 “Fermis”)

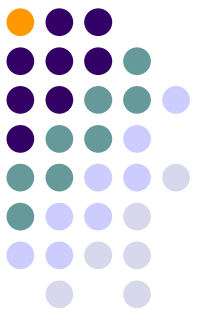
Cost of energy

Saving opportunities



EA1: no polling when there is no work
EA2: no polling when work is in GPU

Performance and energy consumption Summary



- A battle to be won in the core arena
 - More concurrency
 - Heterogeneous designs
- A related battle to be won in the power arena
 - Do nothing, efficiently... (V. Pallipadi, A. Belay)
 - Don't forget the cost of uncore power

...but don't always believe the salesman!

