www.bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Reducing power consumption through dynamic load balancing

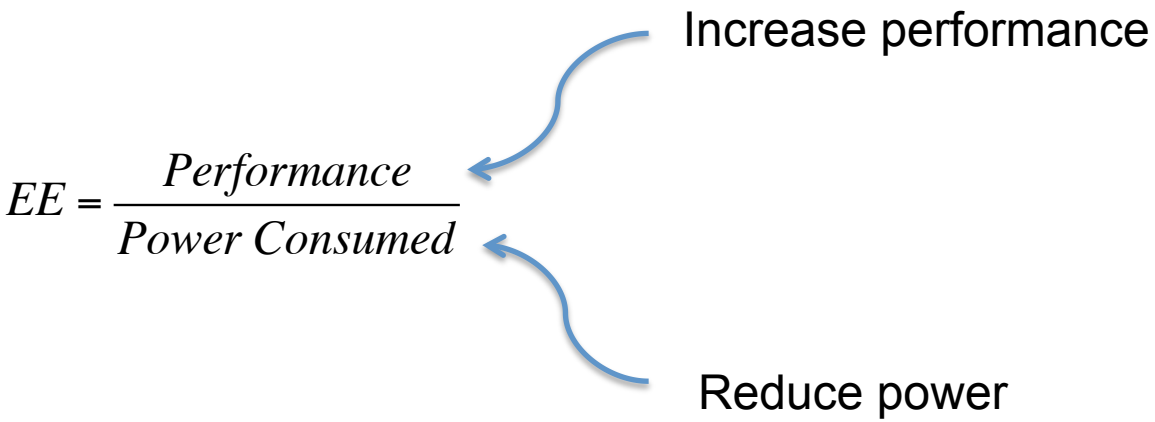
Rosa M. Badia, Jesús Labarta, Marta Garcia, Victor Lopez
BSC

Teratec 2013, Paris, June 26th 2013

Outline

- Energy efficiency and sources of imbalance
- OmpSs overview
- How to measure energy in OmpSs applications
- DLB and LeWI
- Load imbalance analysis with CGPOP
- Conclusions

Energy efficiency

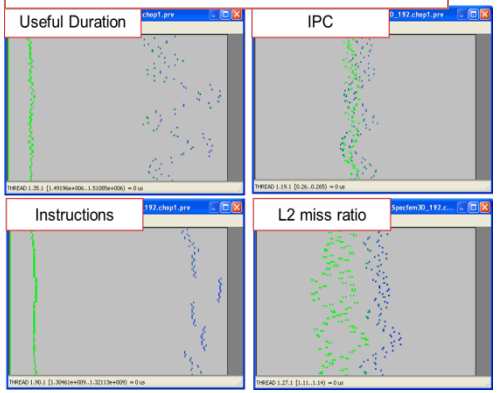


Sources of loose of performance

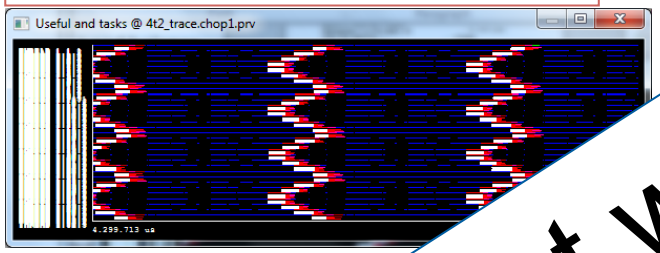
- Load imbalance
- Synchronization
- Resource contention
- Inability to adapt to variability
- …

Load imbalance is everywhere

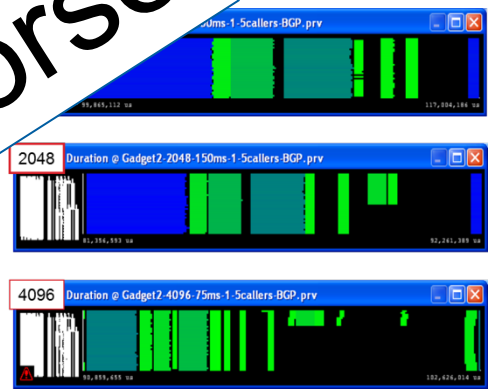
Old SPECEFEM3D version



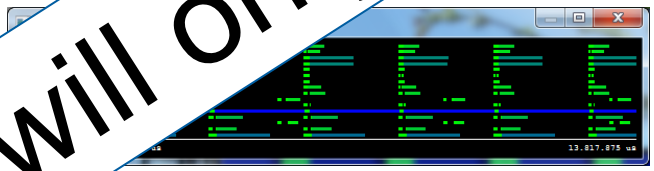
PSC @ 32x4 cores MPI/SMPSs



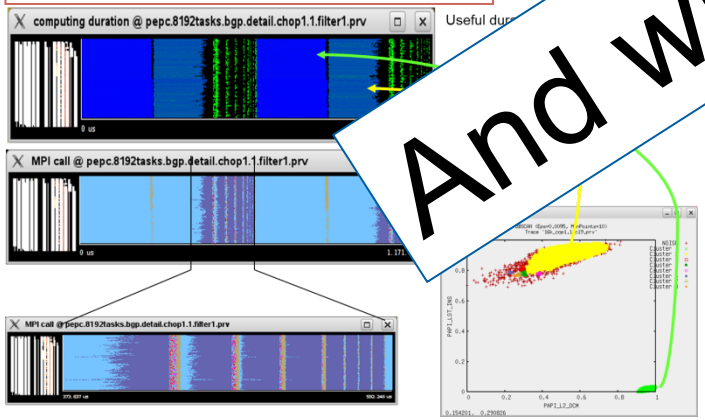
1024 - 4096 cores



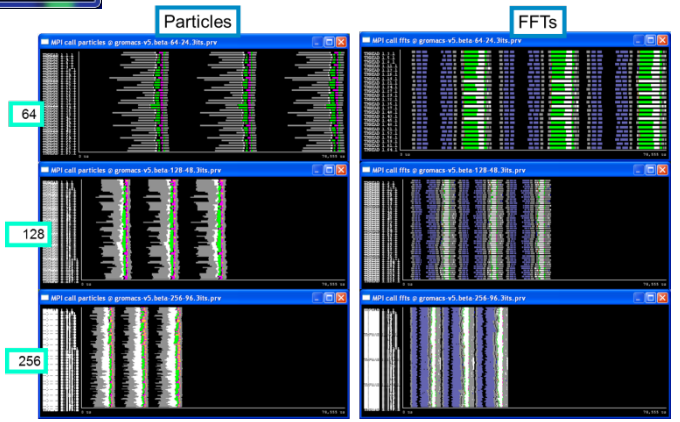
LS1 @



Original PEPC @ 8K cores



Gromacs @ 64-265 cores



And will only get worse

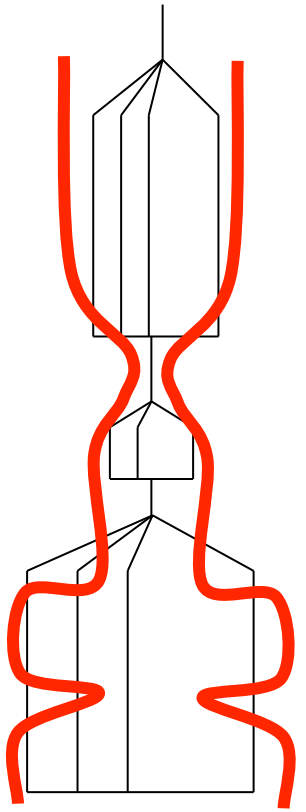
Our believe

- Fighting variability is a lost battle
  - We must learn to live with it
- Malleability is key !!!
  - We must learn how to adapt

> Be water, my friend !!!
>
> Bruce Lee

Malleability

- Flexible (dynamic) parallelization structure of an application
  - Allows responsiveness to dynamic characteristics of a computation and resource availability
- Malleability requires
  - Separation between
    - Algorithm: Problem logic. Programmer responsibility
    - Scheduling: → efficiency. System responsibility (hints may help)
  - Frequent control points:
    - OpenMP limitation to changes between parallels
- Limited if computation explicitly tied to address space
  - i.e. MPI, CAF, …
- Issue of programming model support and programming practices

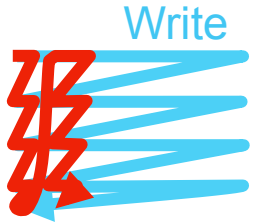


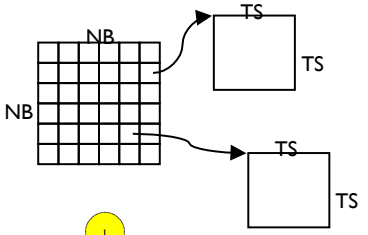
OmpSs programming model

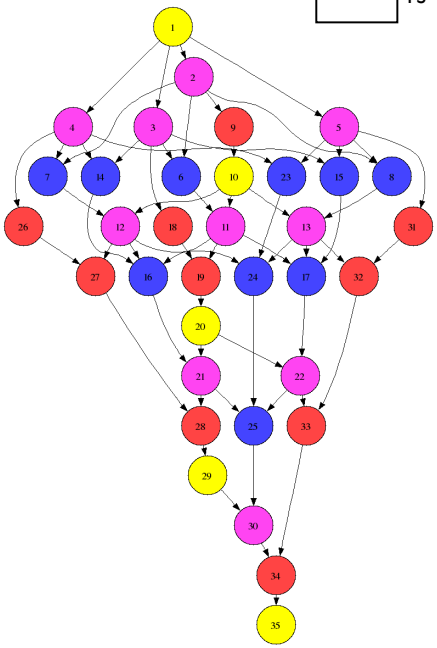
- Sequential program …
  - Task based program on single address/name space
    - Order IS defined !!!!
  - Directionality annotations
    - Used to **compute dependences** AND to provide **data access information**
    - Use pattern, NOT resources and forcing actions (copies,…)
- … happens to execute in parallel
  - Automatic run time computation and honoring of dependencies

OmpSs data-flow execution of sequential programs

Decouple
how we write
form
how it is
executed







```
void Cholesky( float *A ) {
   int i, j, k;
   for (k=0; k<NT; k++) {
      spotrf (A[k*NT+k]) ;
      for (i=k+1; i<NT; i++)
         strsm (A[k*NT+k], A[k*NT+i]);
      // update trailing submatrix
      for (i=k+1; i<NT; i++) {
         for (j=k+1; j<i; j++)
            sgemm( A[k*NT+i], A[k*NT+j], A[j*NT+i]);
         ssyrk (A[k*NT+i], A[i*NT+i]);
```

```
#pragma omp task inout ([TS][TS]A)
void spotrf (float *A);
#pragma omp task input ([TS][TS]A) inout ([TS][TS]C)
void ssyrk (float *A, float *C);
#pragma omp task input ([TS][TS]A,[TS][TS]B) inout ([TS]
[TS]C )
void sgemm (float *A, float *B, float *C);
#pragma omp task input ([TS][TS]T) inout ([TS][TS]B)
void strsm (float *T, float *B);
```

OmpSs: Directives

Task implementation for a GPU device
The compiler parses CUDA/OpenCL kernel invocation syntax

Provides configuration for CUDA/OpenCL kernel

```
#pragma omp target device ({ smp | cuda | opencl }) \
        [ndrange (…)]\
        [ implements ( function_name )]        \
        { copy_deps | [ copy_in ( array_spec ,...)] [ copy_out (...)] [ copy_inout (...)] }
```

Support for multiple implementations of a task

To compute dependences

Ask the runtime to ensure data is accessible in the address space of the device

```
#pragma omp task [ in (...)] [ out (...)] [ inout (...)] [ concurrent (...)] [ commutative (…)] [priority(…)]\
 [label(tasklabel)]

{ function or code block }
```

To set priorities to tasks

To relax dependence order allowing concurrent execution of tasks

To relax dependence order allowing change of order of execution of commutative tasks

```
#pragma omp taskwait [on (...)] [noflush]
```

Wait for sons or specific data availability

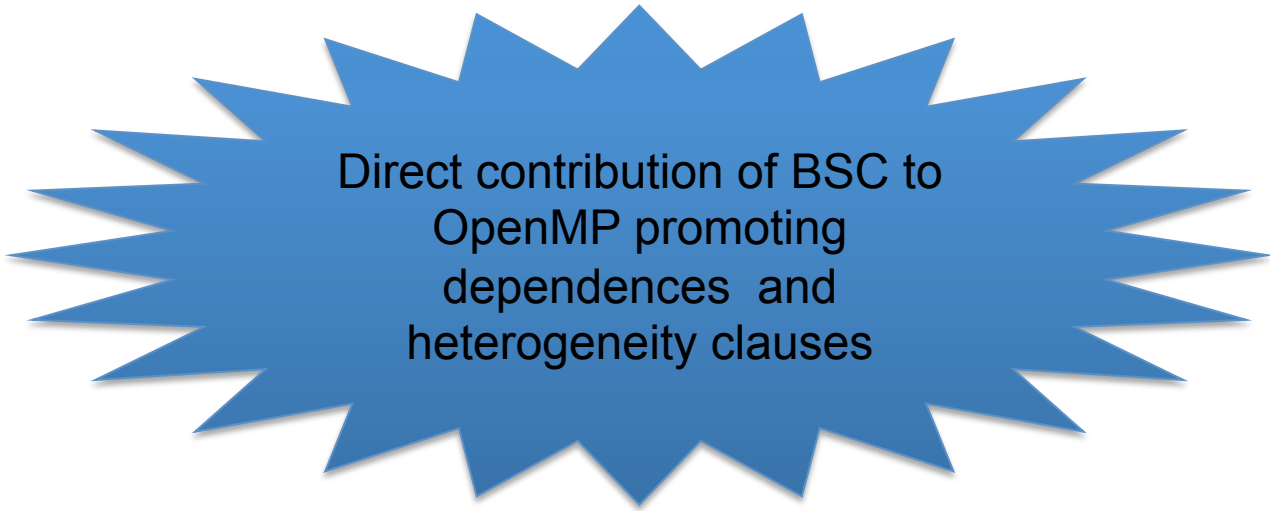
Relax consistency to main program

OpenMP: Directives

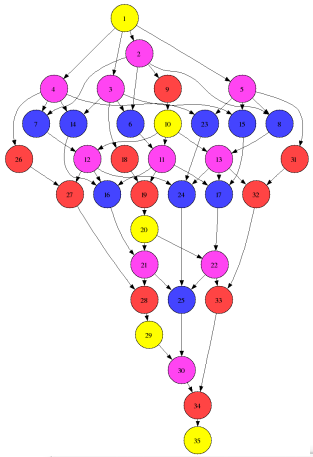
OpenMP dependence specification

#pragma omp task [depend (in: …)] [depend(out:…)] [depend(inout:...)]

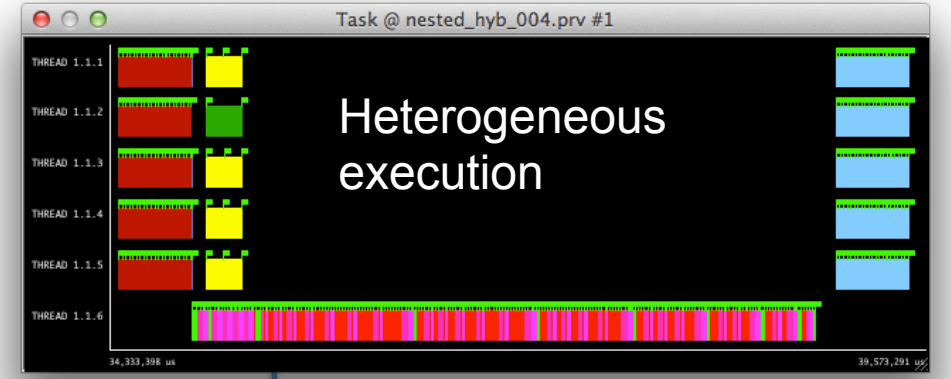
{ function or code block }



OmpSs at execution

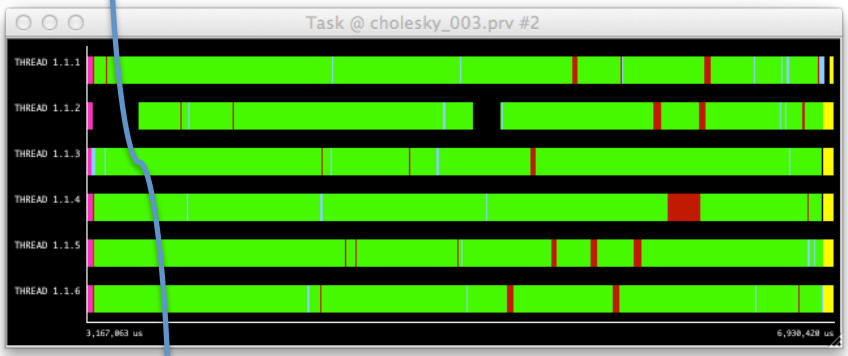


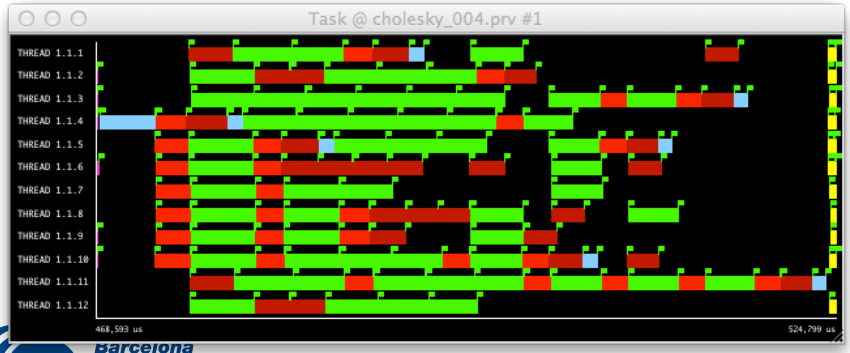
Cholesky

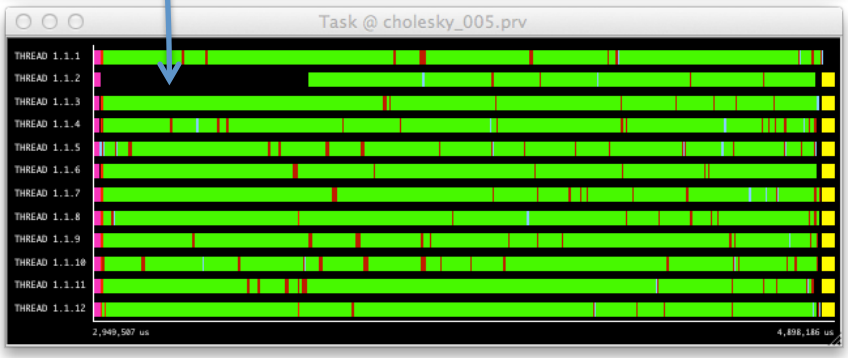


Small problem size

Larger problem size







socket aware scheduling @ OmpSs

- Assigns tasks to a given NUMA node at task creation
  - nested tasks will be assigned to the same node as their parent
- Queues sorted by priority with as many queues as NUMA nodes specified

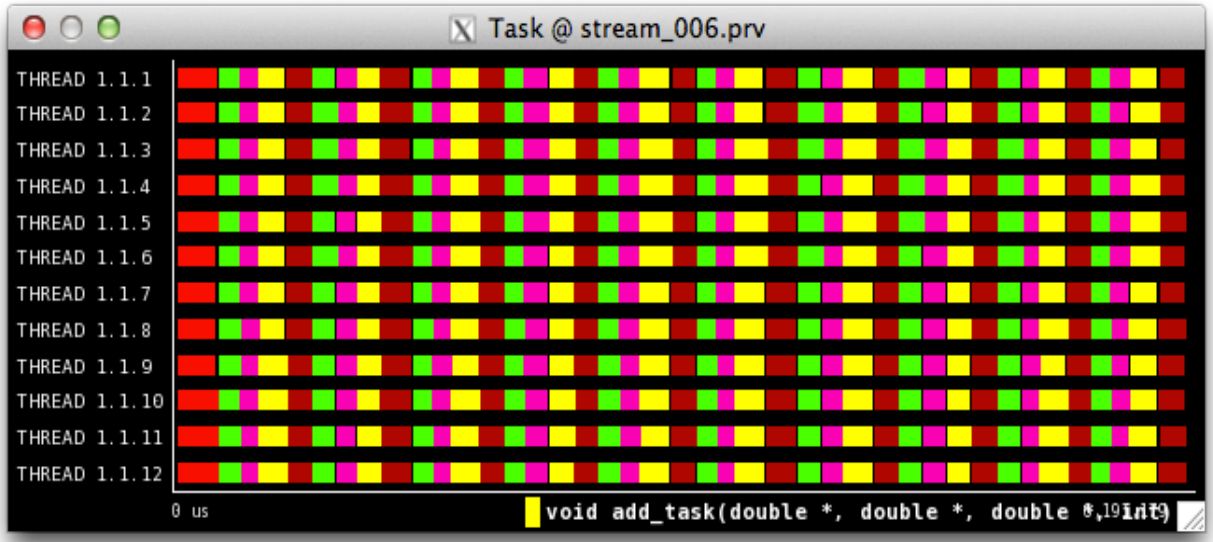
```
#pragma omp task in ([bs]a, [bs]b) out ([bs]c)
void add_task (double *a, double *b, double *c, int bs)
{
    int j;
    for (j=0; j < BSIZE; j++)
       c[j] = a[j]+b[j];
}

void tuned_STREAM_Add()
{
    int j;
    for (j=0; j<N; j+=BSIZE){
        nanos_current_socket( ( j/((int)BSIZE) ) % 2  );
        add_task(&a[j], &b[j], &c[j], BSIZE);
    }
}
```

Currently more invasive that we would like

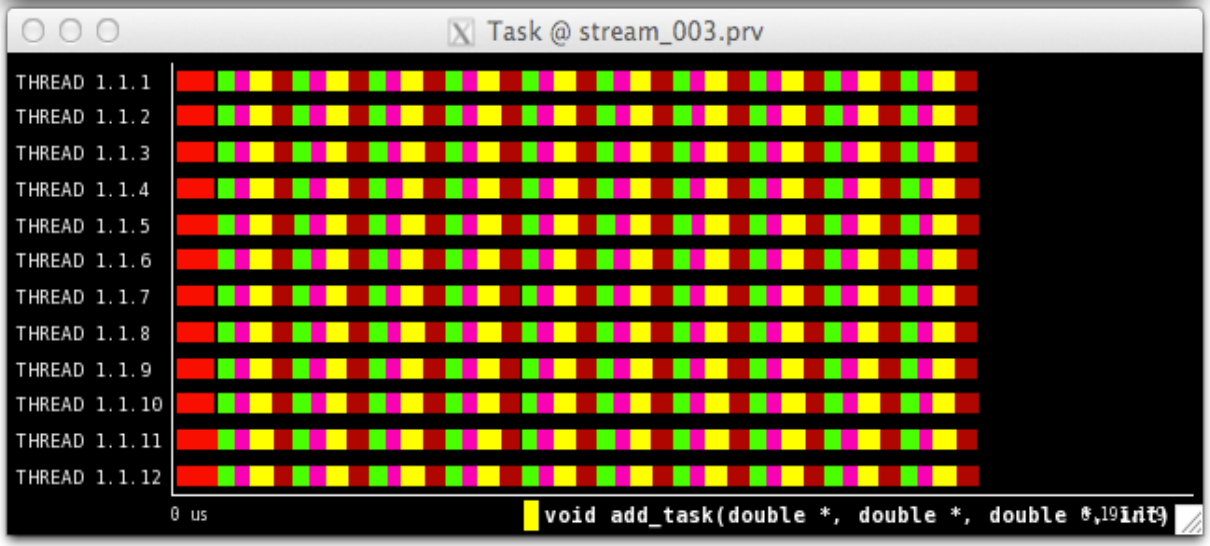
socket aware scheduling @ OmpSs

Stream benchmark

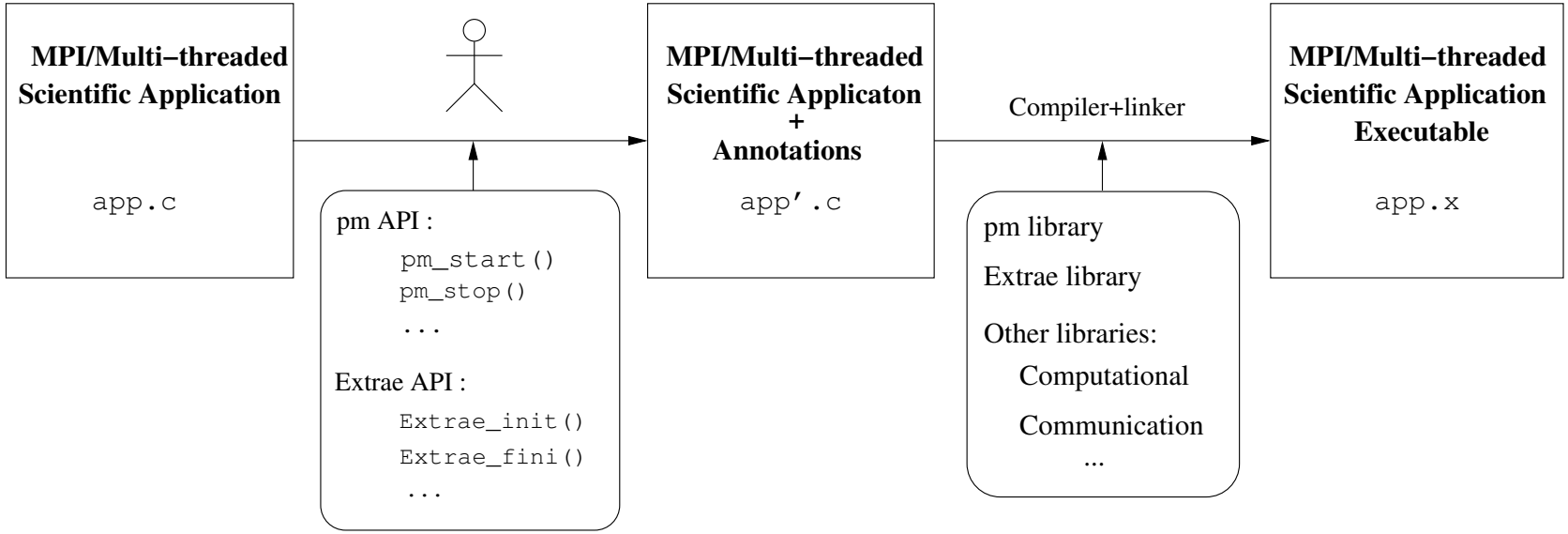
Non Socket-aware

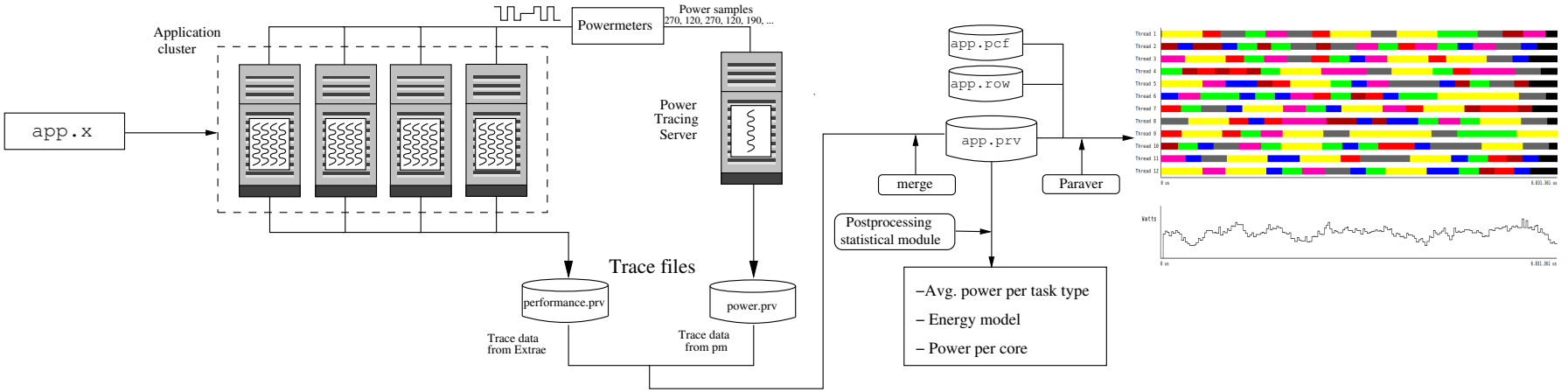


Socket-aware



Integrated framework for power and energy analysis



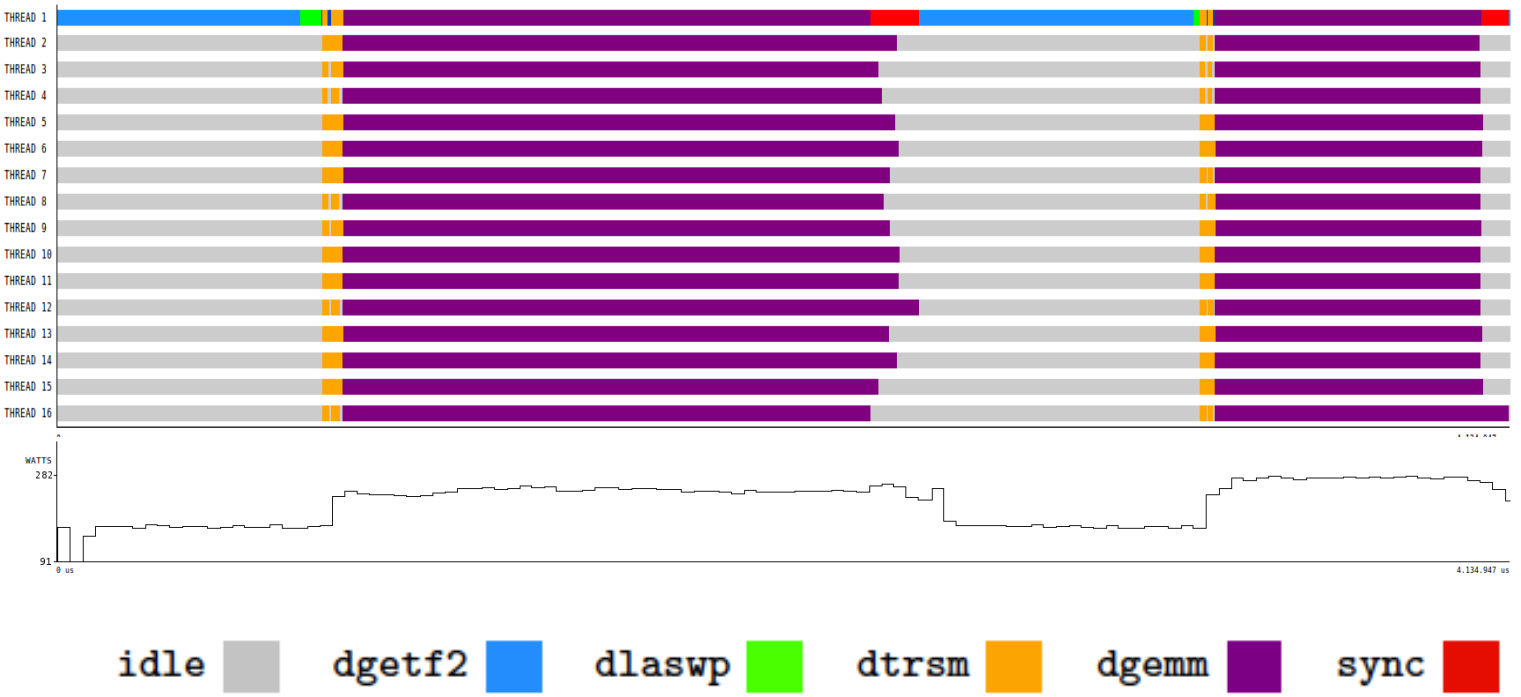


"Tools for Power and Energy Analysis of Parallel Scientific Applications". P. Alonso et al. ICPP12

Integrated framework for power and energy analysis

Kernels

Power

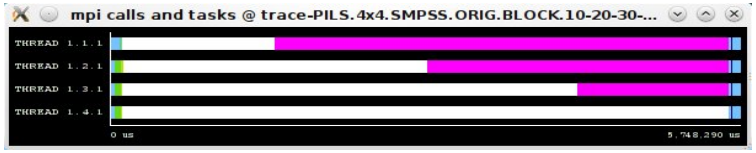


- Using the information in the traces an energy model has been derived for task-based applications
- The environment can be leveraged to write more energy-aware applications
- Can be integrated as a power/energy-aware scheduler in the OmpSs runtime

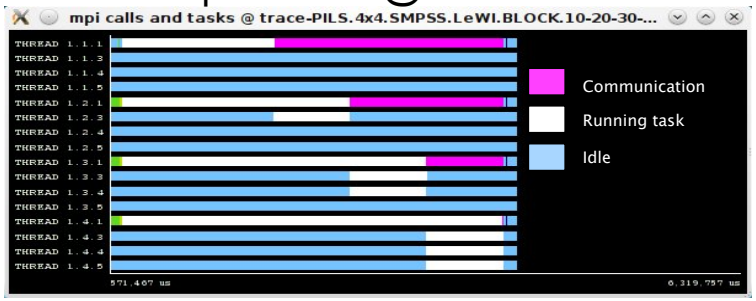
Dynamic Load Balancing: LeWI

- Automatically achieved by the runtime
  - Load balance within node
  - Fine grain
  - Complementary to user level load balance
  - **Leverage OmpSs malleability**

- LeWI: Lend core When Idle
  - User level Run time Library (DLB) coordinating processes within node
  - Lend cores to other processes in the node when entering blocking MPI calls
  - Retrieve when leaving blocking MPI
  - Fighting the Linux kernel: Explicit pinning of threads to cores



4 MPI processes @ 4 cores node

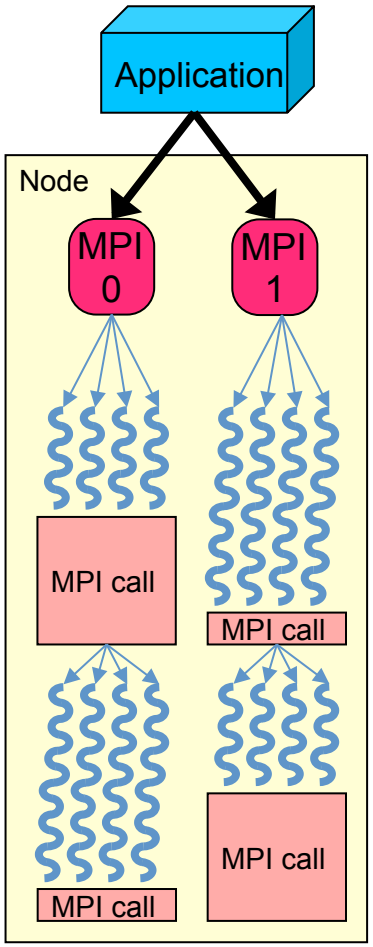


"LeWI: A Runtime Balancing Algorithm for Nested Parallelism". M.Garcia et al. ICPP09

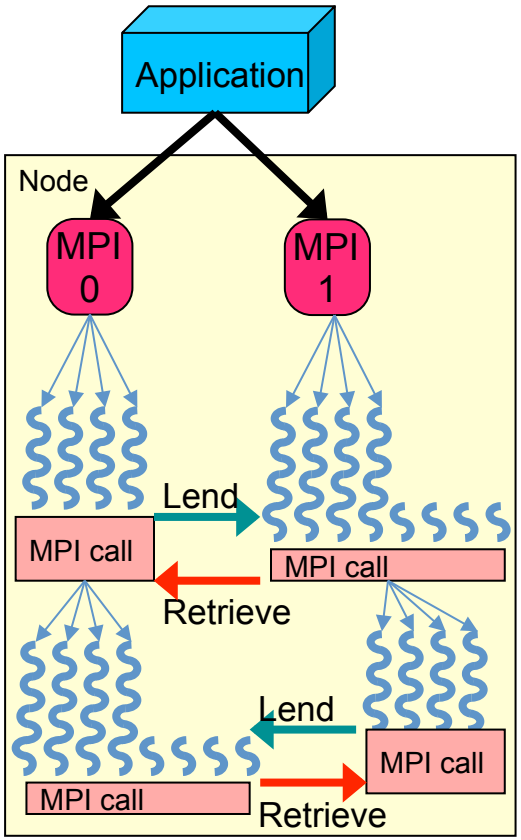
LeWI Load Balancing

- **LeWI**: Lend When Idle
  - An MPI process lends its cpus when inside a blocking MPI call.
  - Another MPI process in the same node can use the lended cpus to run with more threads.
  - When the MPI call is finished the MPI process retrieves it's cpus
- Can handle irregular imbalance
- Its performance depends on the malleability of the second level of parallelism...

- Unbalanced application

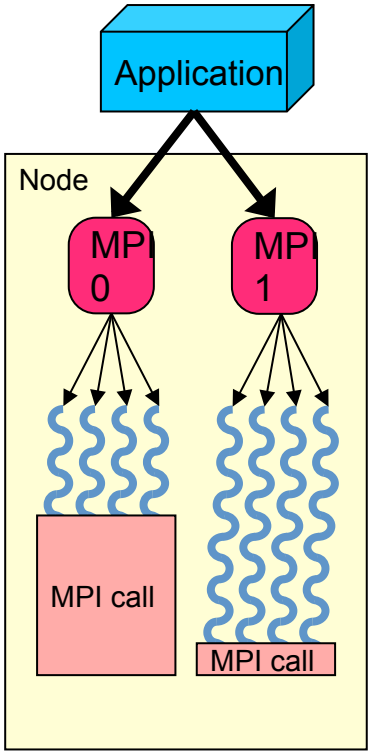


- Execution with DLB

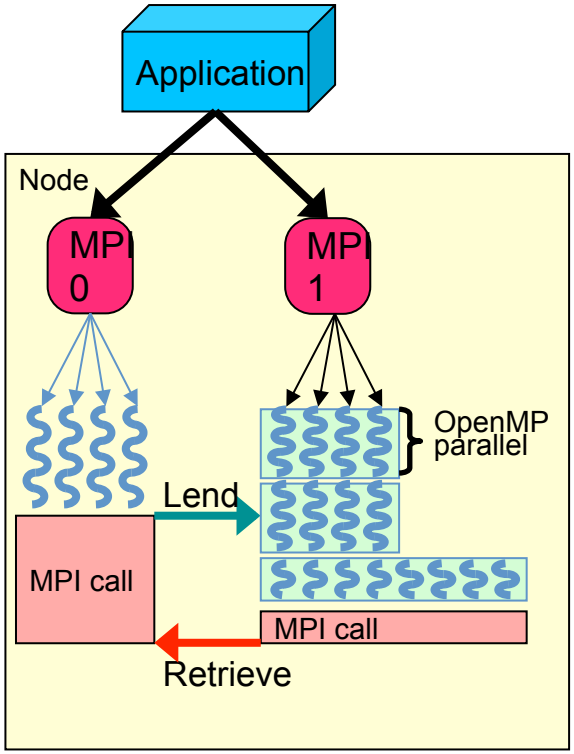


LeWI - Malleability

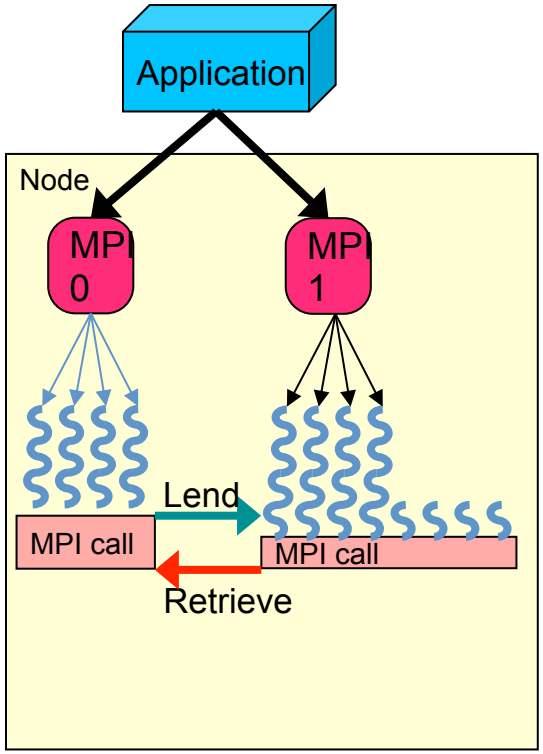
- Unbalanced application



- Execution with DLB MPI+OpenMP



- Execution with DLB MPI+OmpSs



Fighting the OS Scheduler

SMPSs

- Several threads running in the same core (same color)
- Not overloading the node
  - OS Scheduler decision

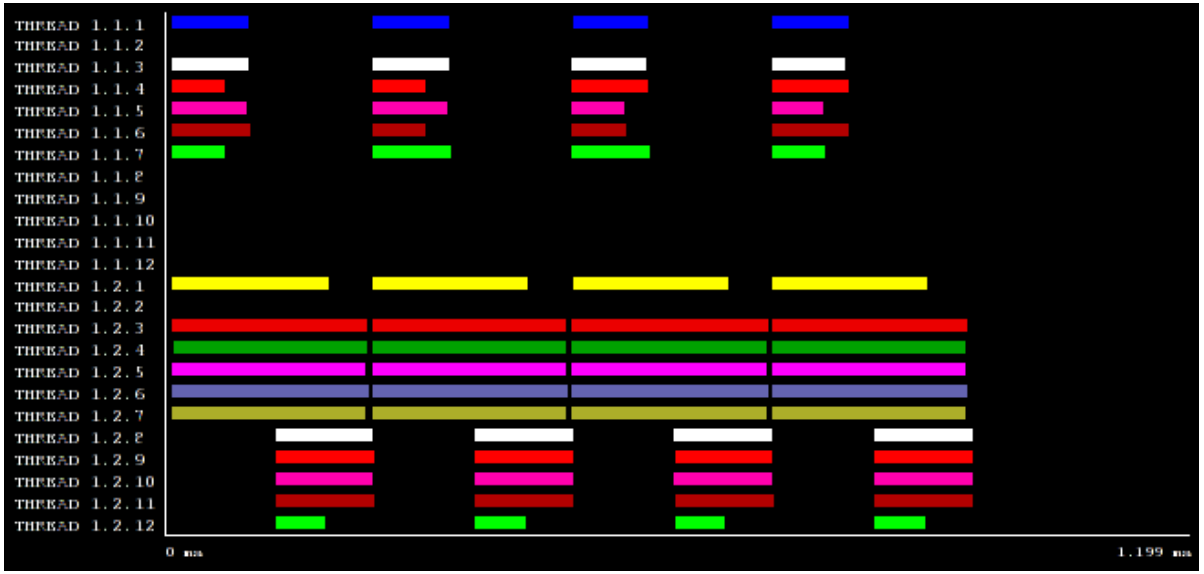
Fighting the OS Scheduler

SMPSs

- Several threads running in the same core (same color)
- No overloading the node
  - OS Scheduler decision

Pinning of threads

- Lend specific cpus
  - Bind threads to cpus
- Cores used exclusivelly
  - No preemptions



Tunning the LeWI policy

- When to lend CPU
  - In any MPI blocking call
  - Only in barrier
- Greedy?
  - Take all the available cpus
  - Take one by one
  - Take only as many cpus as tasks ready to run
- Who takes the cpus?
  - First come First served
  - CPU Affinity (socket aware)

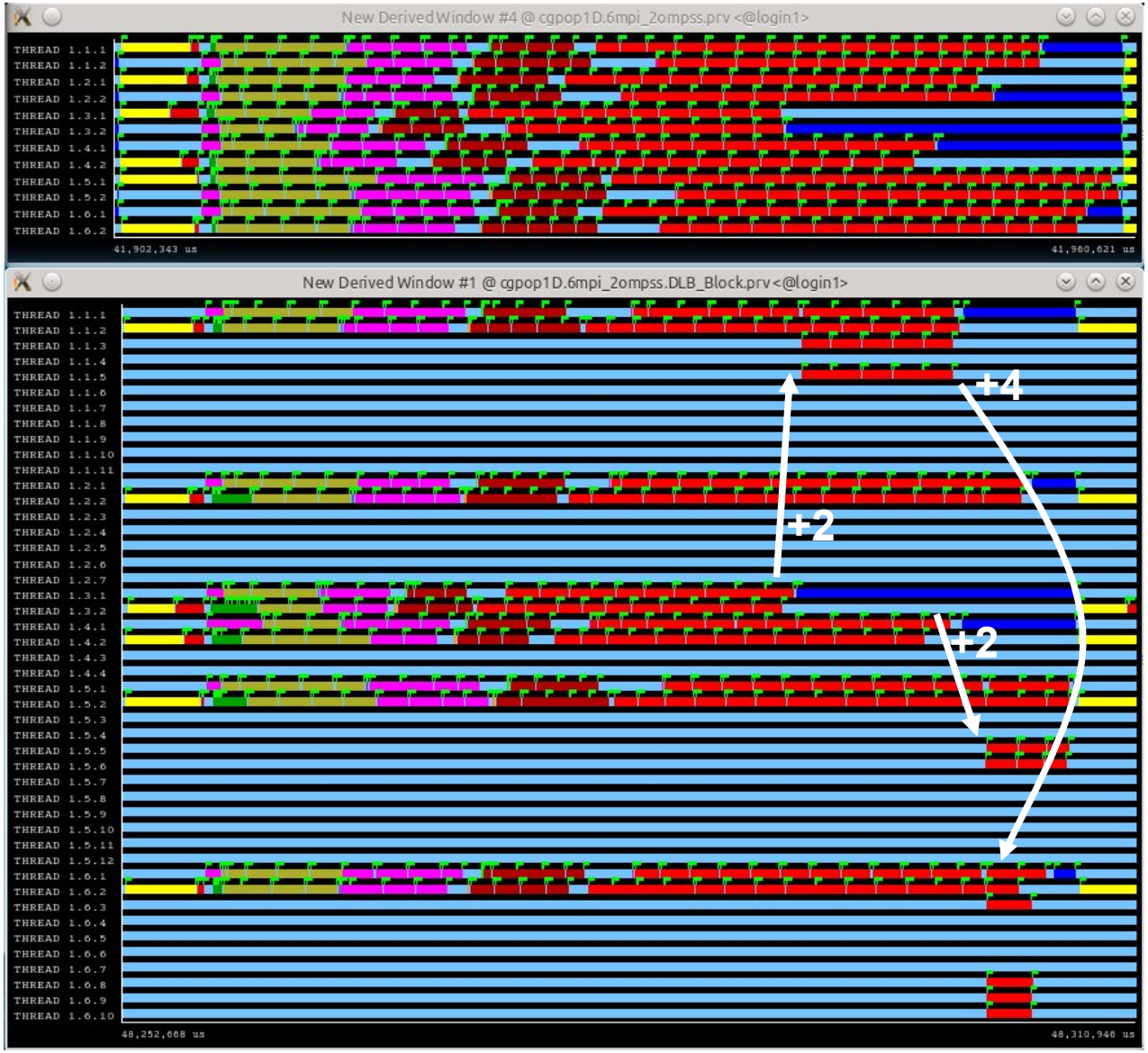
DLB in action

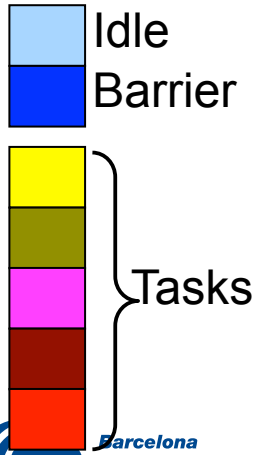
Original

- 6 MPIs
- 2 threads x MPI

DLB

- 6 MPIs
- Initial 2 threads x MPI
- Up to 12 threads x MPI
- 12 threads active at most





Barcelona Supercomputing Center
Centro Nacional de Supercomputación

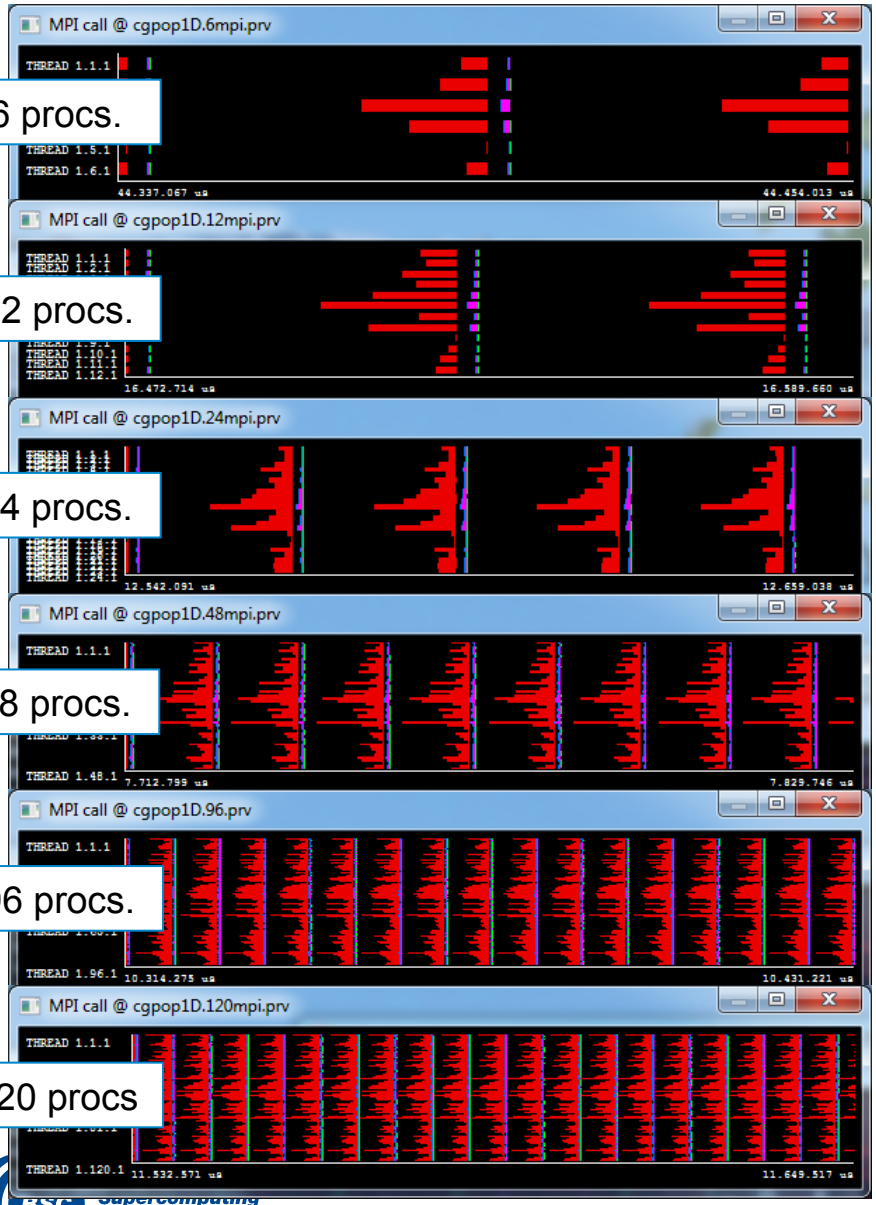
22

CGPOP @ MPI

```
iter_loop: do m = 1, solv_max_iters
  sumN1=c0
  sumN3=c0
  do i=1,nActive

    Z(i) = Minv2(i)*R(i)
    sumN1 = sumN1 + R(i)*Z(i)
    sumN3 = sumN3 + R(i)*R(i)
  enddo

  call matvec(n,A,AZ,Z)


     sumN2=c0
     do i=1,nActive
         sumN2 = sumN2 + AZ(i)*Z(i)
     enddo
  call reduce_update_halo(AZ)
  do i=1,n

       stmp = Z(i) + cg_beta*S(i)
       X(i) = X(i) + cg_alpha*stmp
       S(i) = stmp
  enddo
  do i=1,n

       qtmp = AZ(i) + cg_beta*Q(i)
       R(i) = R(i) - cg_alpha*qtmp
       Q(i) = qtmp
  enddo
end do iter_loop
```

- Fortran + MPI code

Analysis on Intel 12 core nodes

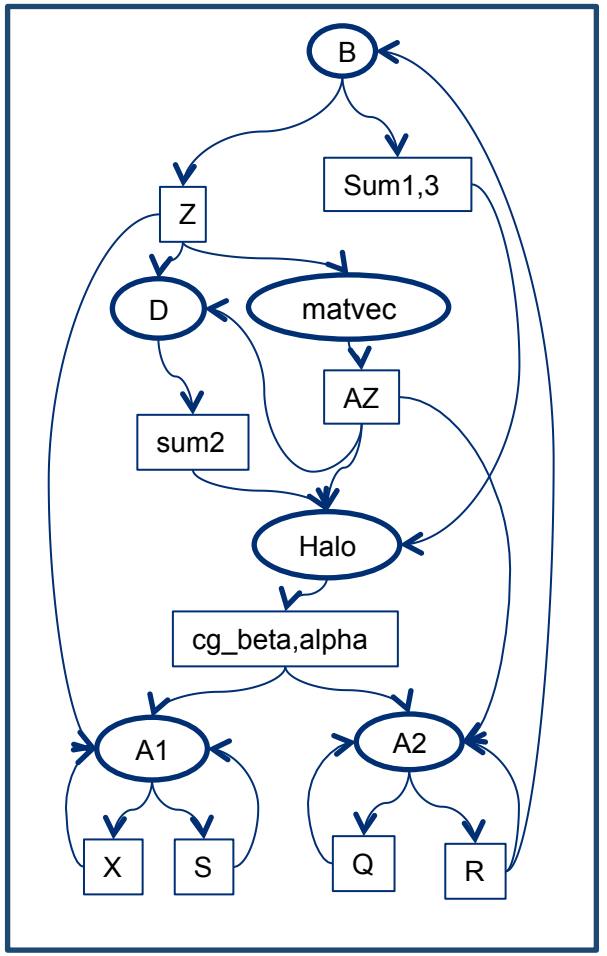


Procs.	Sup	Time LB
6	1.00	0.85
12	1.10	0.85
24	2.20	0.85
48	4.23	0.80
96	7.97	0.70
120	9.74	0.68

- Non linear scaling
- Load Balance, not only not very good but …horrible?
- Potential gain? 15% within node?

CGPOP @ MPI + OmpSs

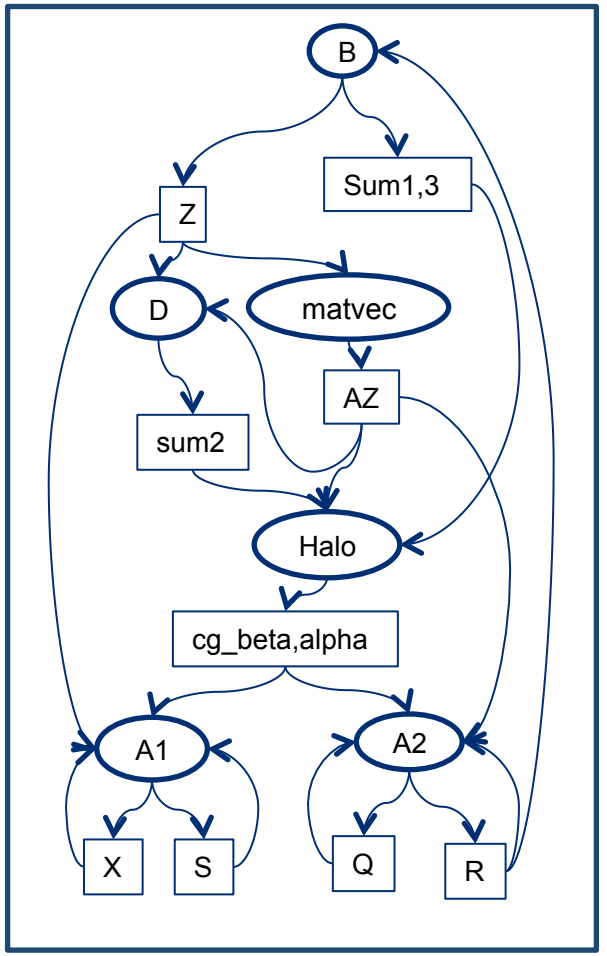
```
iter_loop: do m = 1, solv_max_iters
  sumN1=c0
  sumN3=c0
  do i=1,nActive
!omp task in(R) out (Z) inout (sums)
    Z(i) = Minv2(i)*R(i)
    sumN1 = sumN1 + R(i)*Z(i)
    sumN3 = sumN3 + R(i)*R(i)
  enddo
!omp taskwaiton (Z)
  call matvec(n,A,AZ,Z)

  !omp task in(Z,AZ) inout(sum2)
     sumN2=c0
     do i=1,nActive
        sumN2 = sumN2 + AZ(i)*Z(i)
     enddo
  call reduce_update_halo(AZ)
  do i=1,n
    !omp task in(Z) inout(S,X)
       stmp = Z(i) + cg_beta*S(i)
       X(i) = X(i) + cg_alpha*stmp
       S(i) = stmp
  enddo
  do i=1,n
    !omp task in(AZ) inout (Q,R)
       qtmp = AZ(i) + cg_beta*Q(i)
       R(i) = R(i) - cg_alpha*qtmp
       Q(i) = qtmp
  enddo
end do iter_loop
```



- Potential to naturally overlap computation and communication

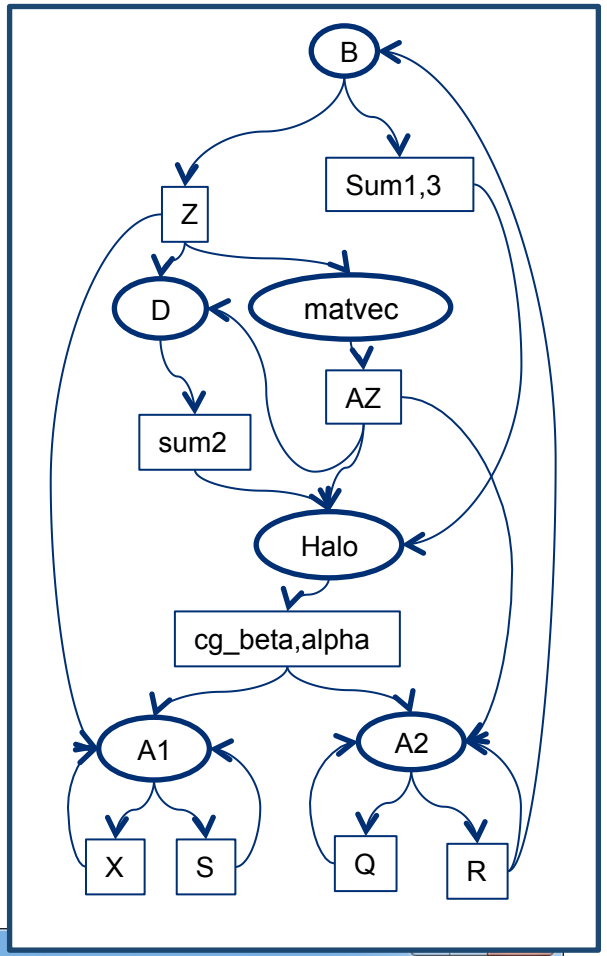
CGPOP @ MPI + OmpSs

```
iter_loop: do m = 1, solv_max_iters
  sumN1=c0
  sumN3=c0
  do i=1,nActive
!omp task in(R) out (Z) inout (sums)
B     Z(i) = Minv2(i)*R(i)
      sumN1 = sumN1 + R(i)*Z(i)
      sumN3 = sumN3 + R(i)*R(i)
  enddo
!omp taskwaiton (Z)
C  call matvec(n,A,AZ,Z)
  Call MPI_barrier()
  !omp task in(Z,AZ) inout(sum2)
D     sumN2=c0
      do i=1,nActive
          sumN2 = sumN2 + AZ(i)*Z(i)
      enddo
  call reduce update halo(AZ)
  do i=1,n
    !omp task in(Z) inout(S,X)
A1        stmp = Z(i) + cg_beta*S(i)
          X(i) = X(i) + cg_alpha*stmp
          S(i) = stmp
  enddo
  do i=1,n
    !omp task in(AZ) inout (Q,R)
A2        qtmp = AZ(i) + cg_beta*Q(i)
          R(i) = R(i) - cg_alpha*qtmp
          Q(i) = qtmp
  enddo
end do iter_loop
```

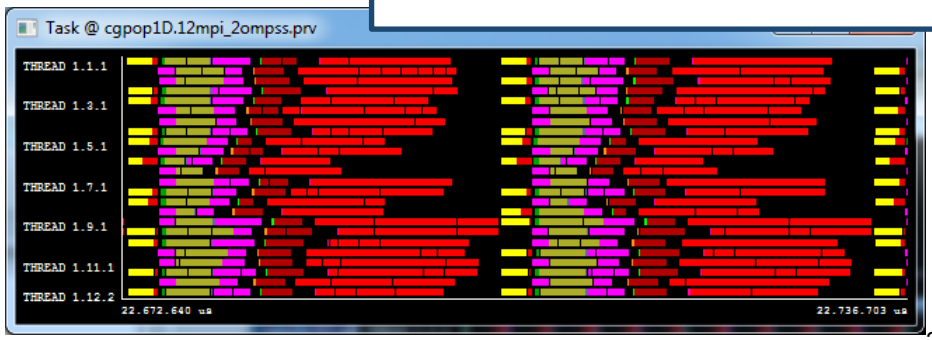


CGPOP @ MPI + OmpSs

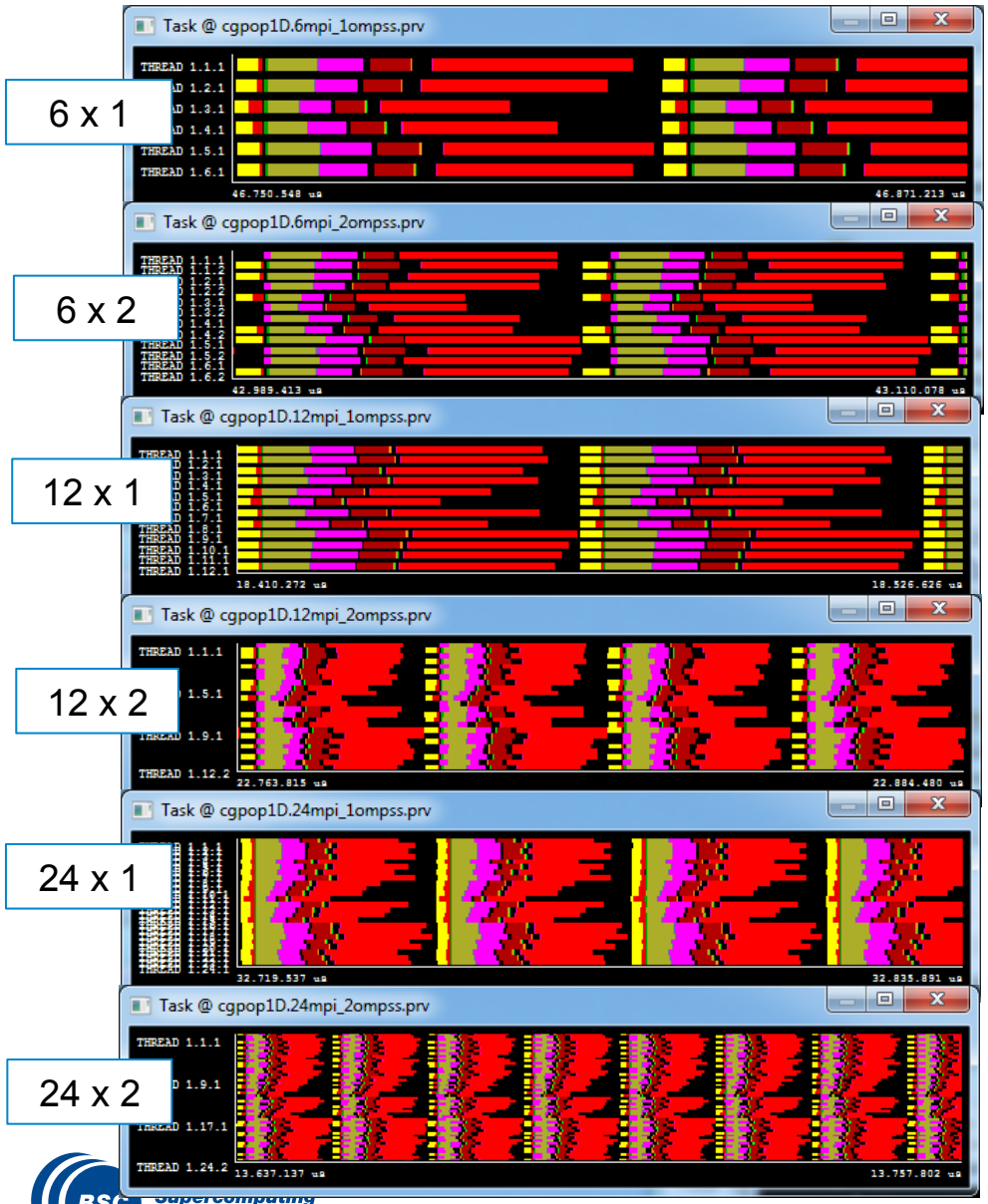
```
iter_loop: do m = 1, solv_max_iters
  sumN1=c0
  sumN3=c0
  do i=1,nActive
!omp task in(R) out (Z) inout (sums)
B     Z(i) = Minv2(i)*R(i)
      sumN1 = sumN1 + R(i)*Z(i)
      sumN3 = sumN3 + R(i)*R(i)
  enddo
!omp taskwaiton (Z)
C  call matvec(n,A,AZ,Z)
   Call MPI_barrier()
D  !omp task in(Z,AZ) inout(sum2)
      sumN2=c0
      do i=1,nActive
          sumN2 = sumN2 + AZ(i)*Z(i)
      enddo
   call reduce update halo(AZ)
   do i=1,n
     !omp task in(Z) inout(S,X)
A1      stmp = Z(i) + cg_beta*S(i)
        X(i) = X(i) + cg_alpha*stmp
        S(i) = stmp
   enddo
   do i=1,n
     !omp task in(AZ) inout (Q,R)
A2      qtmp = AZ(i) + cg_beta*Q(i)
        R(i) = R(i) - cg_alpha*qtmp
        Q(i) = qtmp
   enddo
end do iter_loop
```



12 MPI x 2 Th



Mpi/OmpSs



Pure MPI

Procs.	Sup	Time LB
6	1.00	0.85
12	1.10	0.85
24	2.20	0.85
48	4.23	0.80
96	7.97	0.70
120	9.74	0.68

MPI/OmpSs

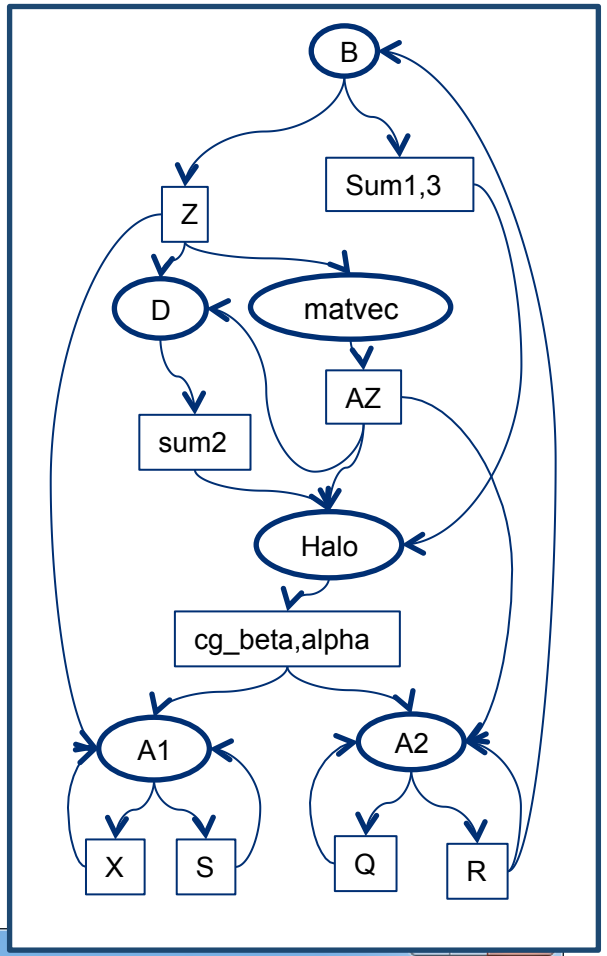
Procs.	Sup	Time LB
6 x 1	0.82	0.85
6 x 2	1.01	0.80
12 x 1	1.06	0.85
12 x 2	1.89	0.80
24 x 1	1.85	0.81
24 x 2	3.65	0.77

- Overhead vs. pure MPI
- Potential gain? 15% within a node?

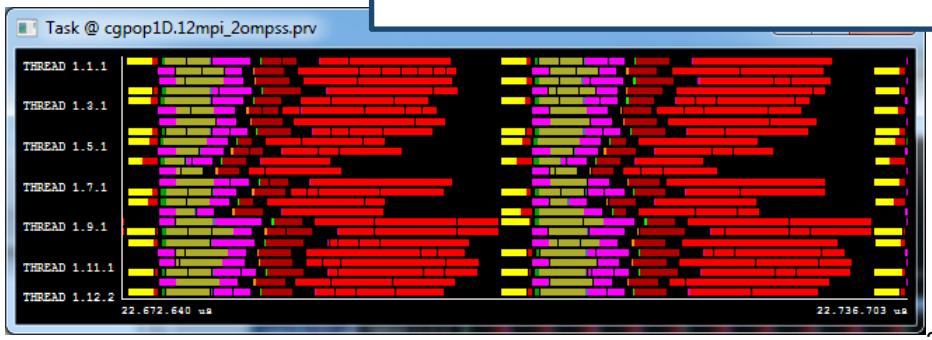
CGPOP @ MPI + OmpSs

```
iter_loop: do m = 1, solv_max_iters
  sumN1=c0
  sumN3=c0
  do i=1,nActive
!omp task in(R) out (Z) inout (sums)
    Z(i) = Minv2(i)*R(i)                    B
    sumN1 = sumN1 + R(i)*Z(i)
    sumN3 = sumN3 + R(i)*R(i)
  enddo
!omp taskwaiton (Z)
  call matvec(n,A,AZ,Z)                     C
  Call MPI_barrier()
  !omp task in(Z,AZ) inout(sum2)
    sumN2=c0                                D
    do i=1,nActive
      sumN2 = sumN2 + AZ(i)*Z(i)
    enddo
  call reduce update halo(AZ)
  do i=1,n
    !omp task in(Z) inout(S,X)
      stmp = Z(i) + cg_beta*S(i)            A1
      X(i) = X(i) + cg_alpha*stmp
      S(i) = stmp
  enddo
  do i=1,n
    !omp task in(AZ) inout (Q,R)
      qtmp = AZ(i) + cg_beta*Q(i)           A2
      R(i) = R(i) - cg_alpha*qtmp
      Q(i) = qtmp
  enddo
end do iter_loop
```

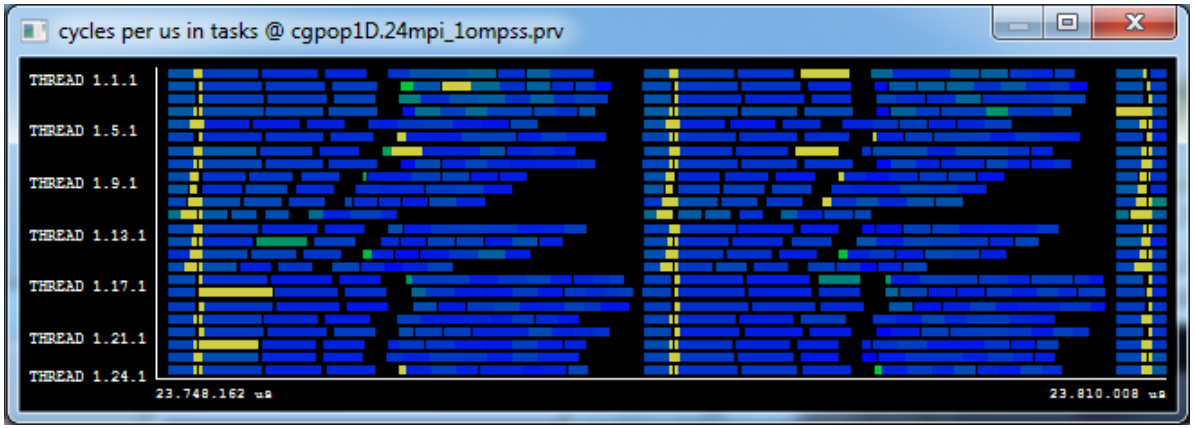
Force DLB



12 MPI x 2 Th

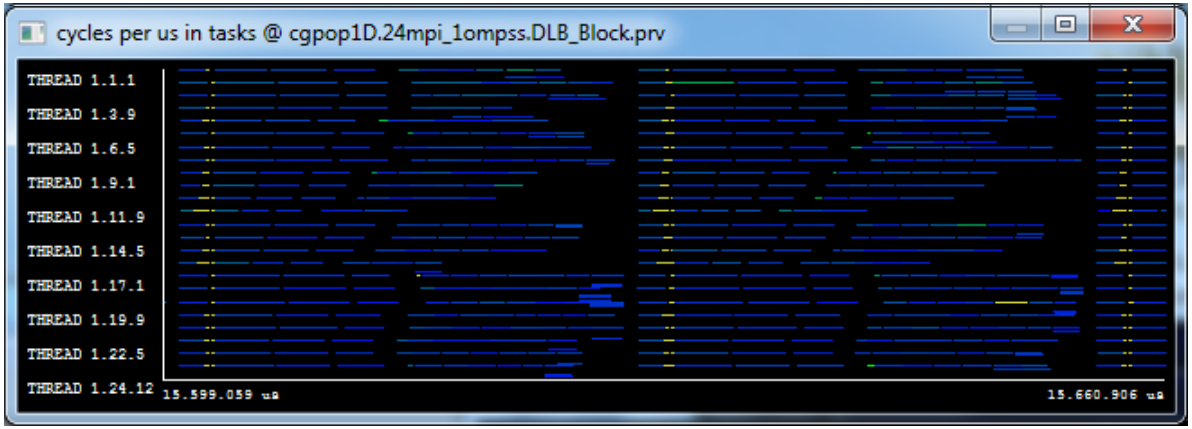


DLB in action ?



MPI/OmpSs
24 MPI 1 Th

- Dynamic reallocation
- Pinned threads



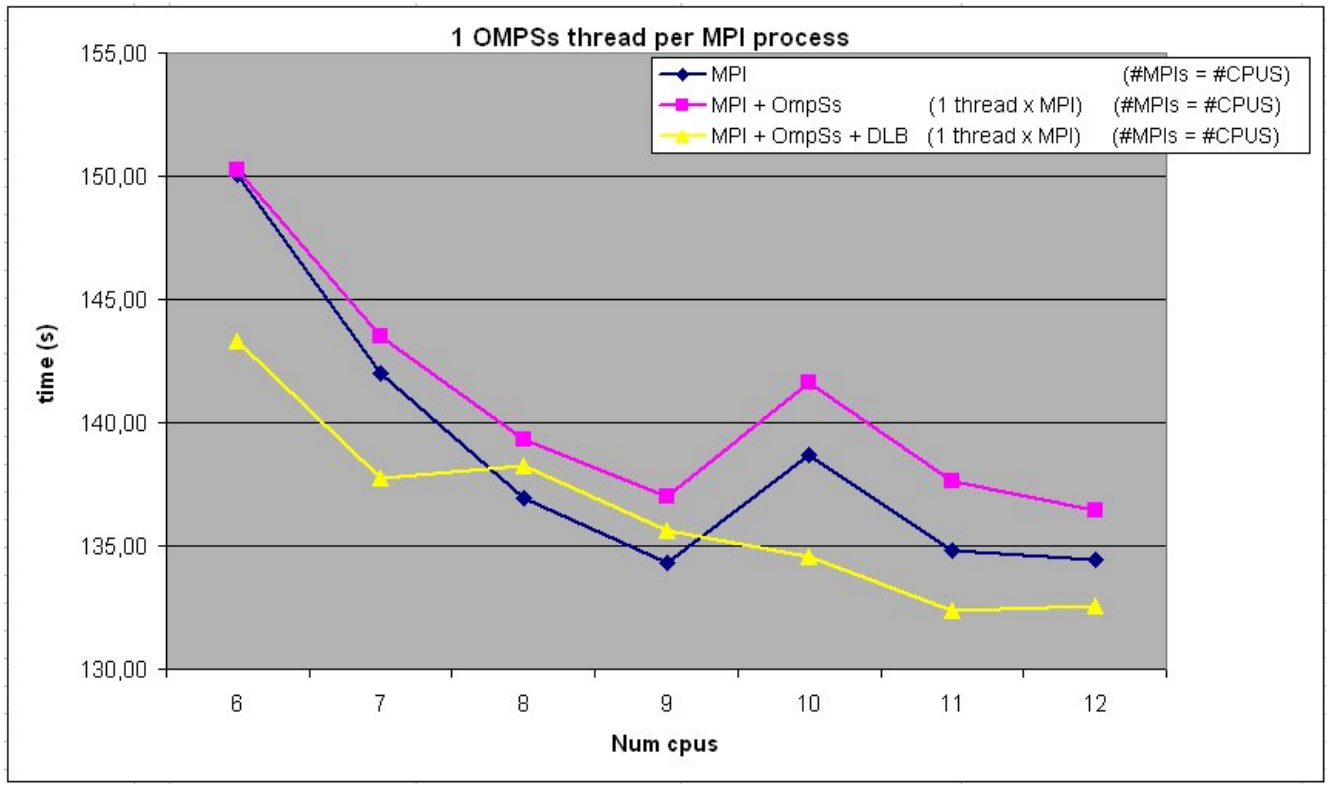
MPI/OmpSs + DLB

- No preemptions

« Effects visible … but timing …

To DLB … or not to DLB

- 6-12 MPIs, 1 threads

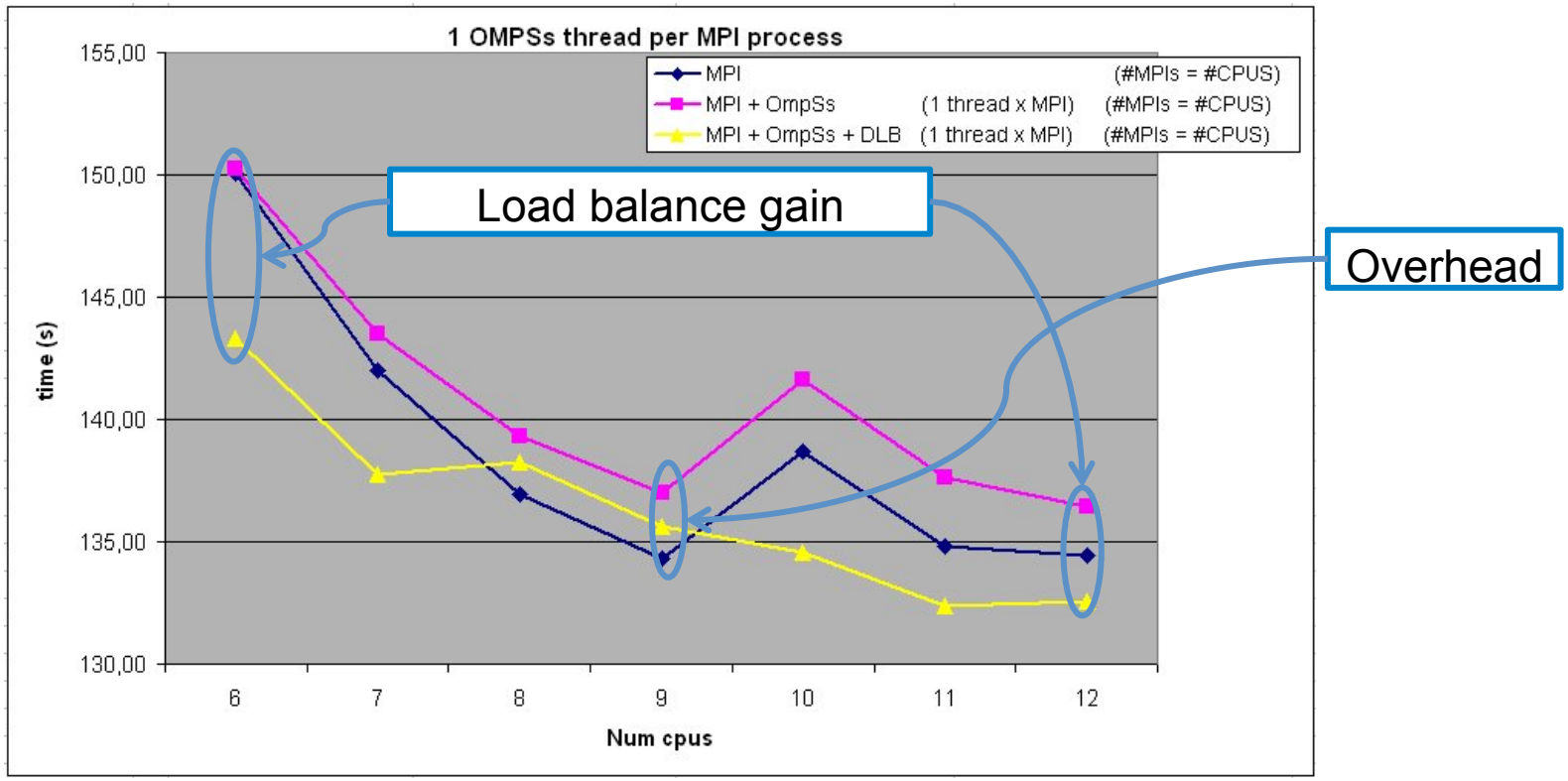


Some gain … not impressive?

By the way, better than instrumented run

To DLB … or not to DLB

- 6-12 MPIs, 1 threads

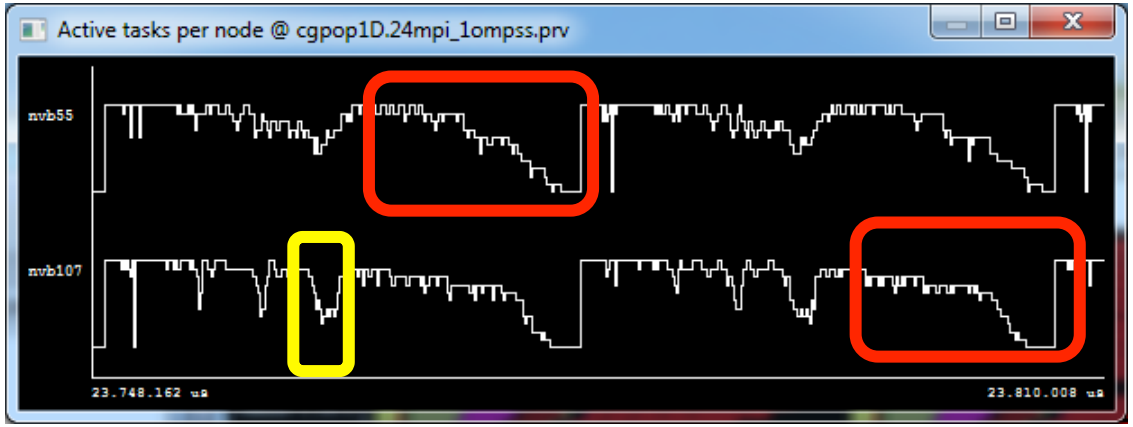


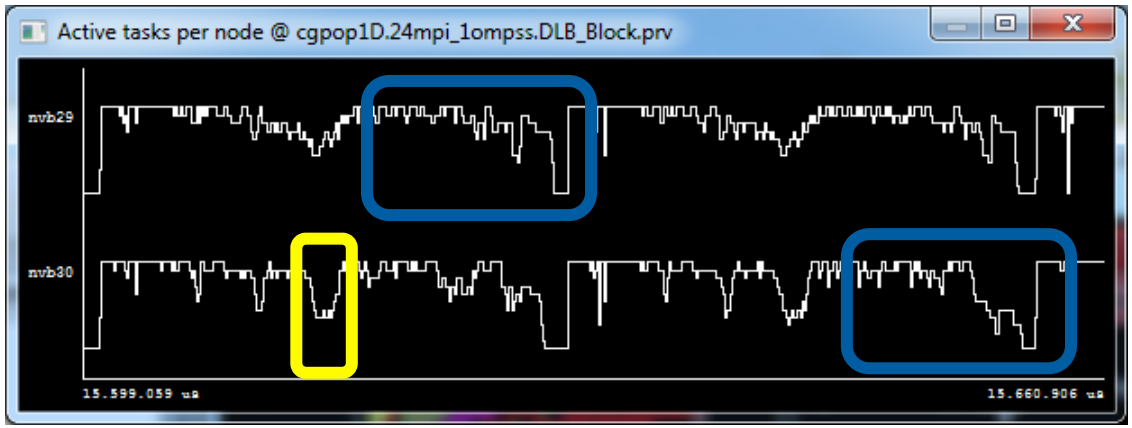
Some gain … not impressive?

By the way, better than instrumented run

DLB in action ?

- Number of tasks executing concurrently per process
  - 24 MPI x 1 thread



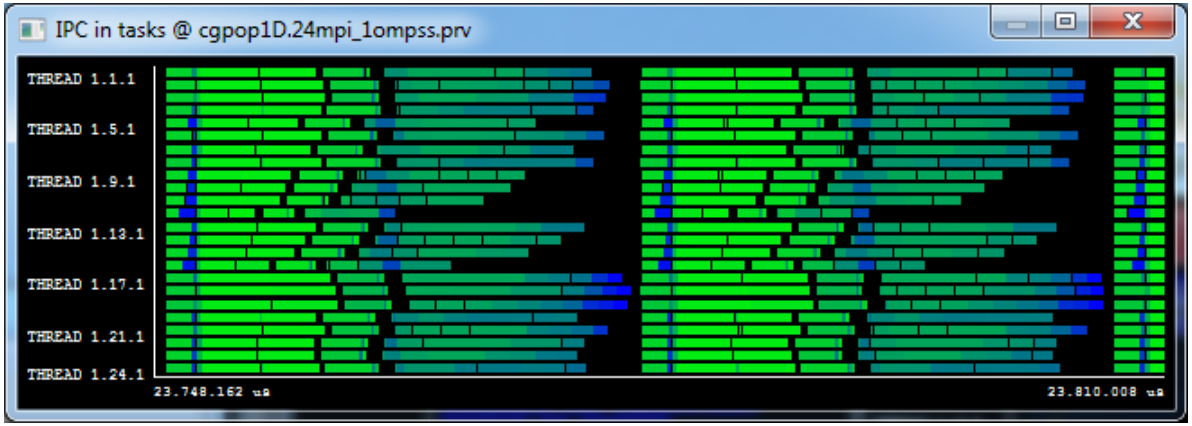


Yes:
- More active threads

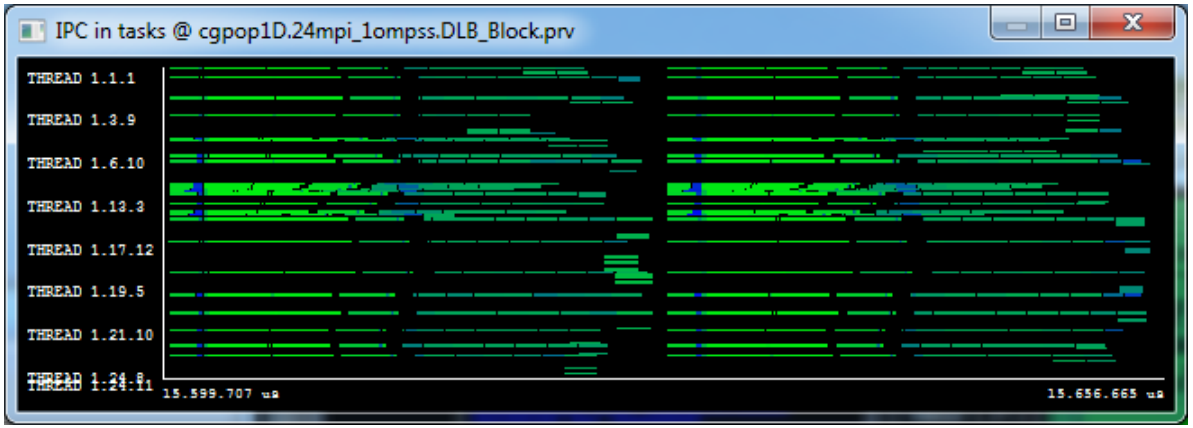
But
- Still some idling …
- …caused by fixed granularity …
- …and task creation overhead

Further analysis

IPC in tasks



Overal IPC fairly poor

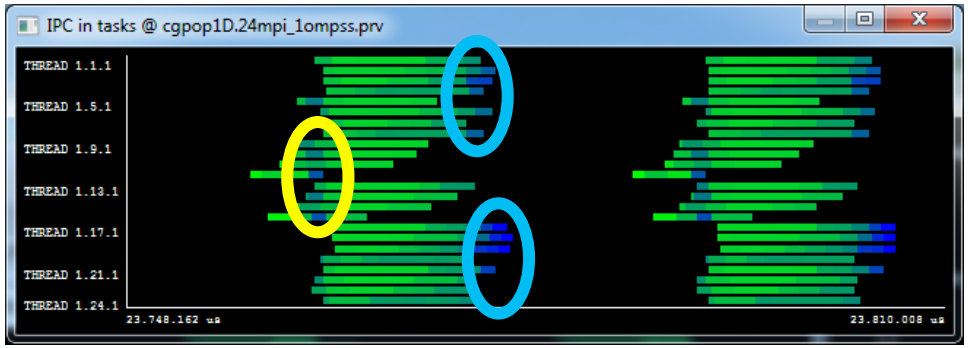


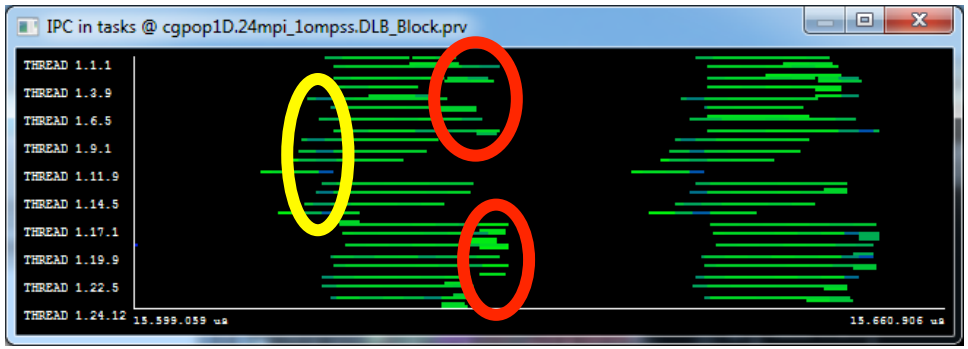
Yes:
- More active threads

But
- Poorer IPC !!!

Further analysis

IPC in matvec tasks





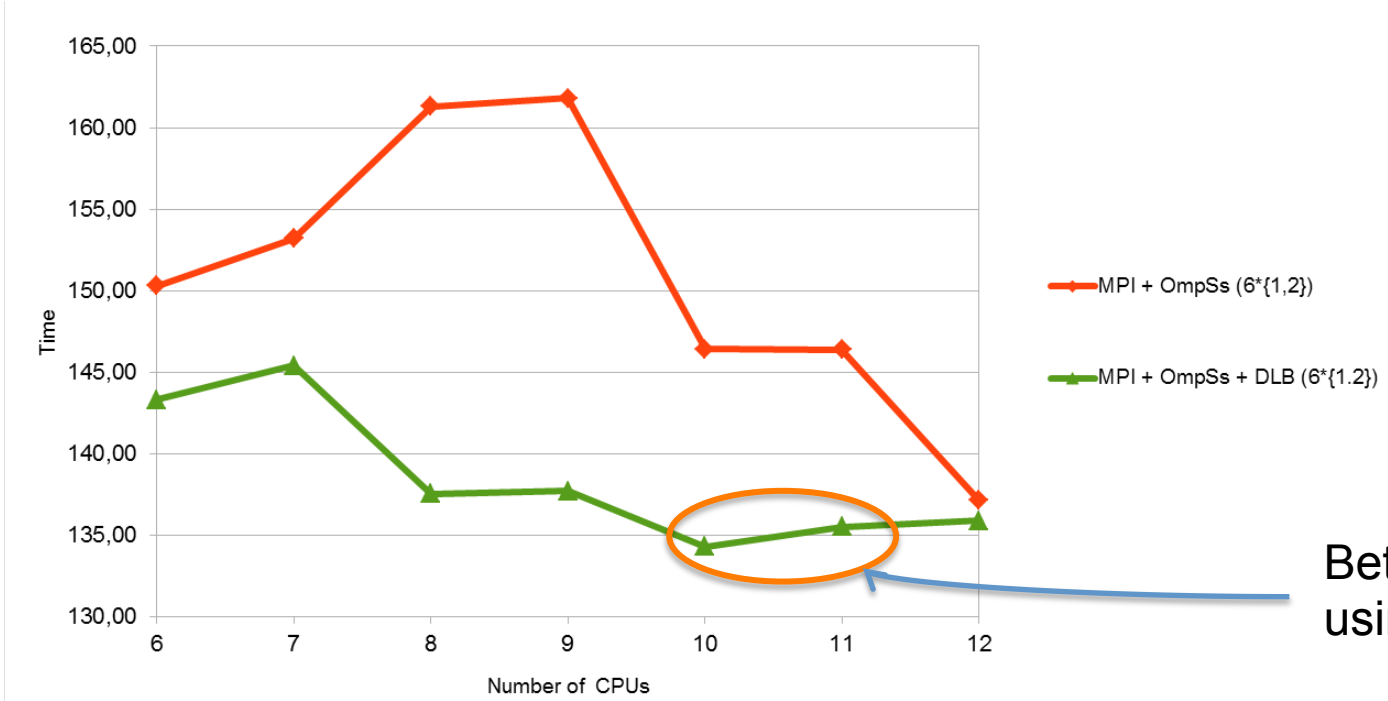
- Less threads →more IPC !!!!
  - Effect visible along matvec execution
- Tasks concurrent with task creation → more IPC !!!

- More threads less IPC !!!!

Memory bandwidth bottleneck → potential less than expected from LB measurement

Non uniform num_threads

- 6 MPIs, 1/2 threads



Better performance using less CPUs

1-1-1-1-1-1 2-1-1-1-1-2 2-2-1-1-2-2 2-2-2-2-2-2
2-1-1-1-1-1 2-2-1-1-1-2 2-2-2-1-2-2

- Were to assign additional cores?
- Potential of dynamic scheduling !!

Conclusions

- Power can be reduced by means of improving performance
- How?
  - Using malleable programming models and runtimes that are able to dynamically react under resource variability: OmpSs
  - Better use of resources, i.e. Memory: Socket aware scheduling @ OmpSs
  - Model applications and integrated power-aware models in runtimes
  - Use of DLB/LeWI and similar runtime tools for load balancing


BSC
Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación
THANKS