Barcelona Supercomputing Center
Centro Nacional de Supercomputación

The OmpSs Programming Model

encore

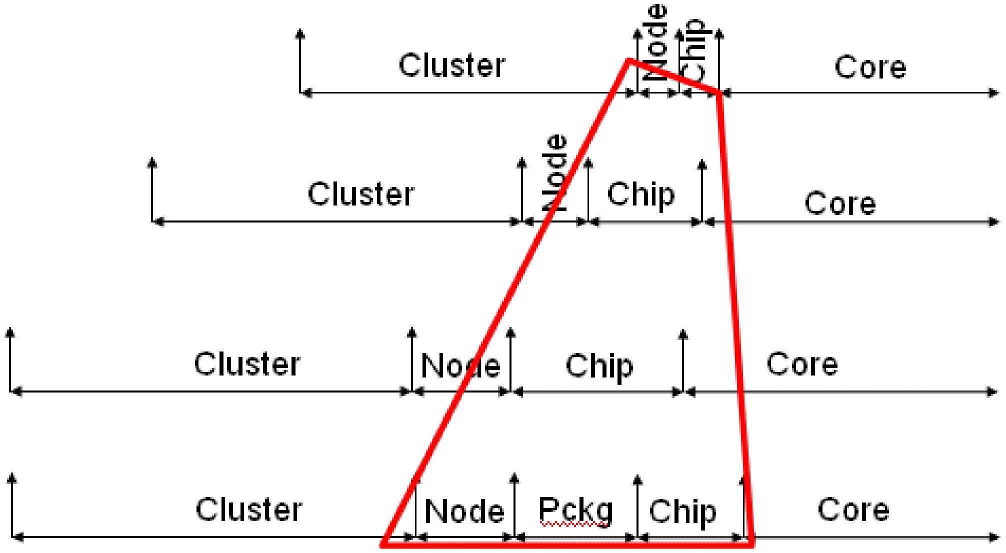
Jesus Labarta
Director Computer Sciences Research Dept.
BSC

Challenges on the way to Exascale

- Efficiency (…, power, …)
- Variability
- Memory
- Faults
- Scale (…,concurrency, strong scaling,…)
- Complexity (…Hierarchy /Heterogeneity,…)

"To kill flies with canons"
Spanish saying

E P T G
1000000000000000000



J. Labarta, et all, "BSC Vision towards Exascale"
IJHPCA vol 23, n. 4 Nov 2009

"The butterfly effect"

Supercomputer Development

Application
Algorithm
Progr. Model
Run time
Architecture

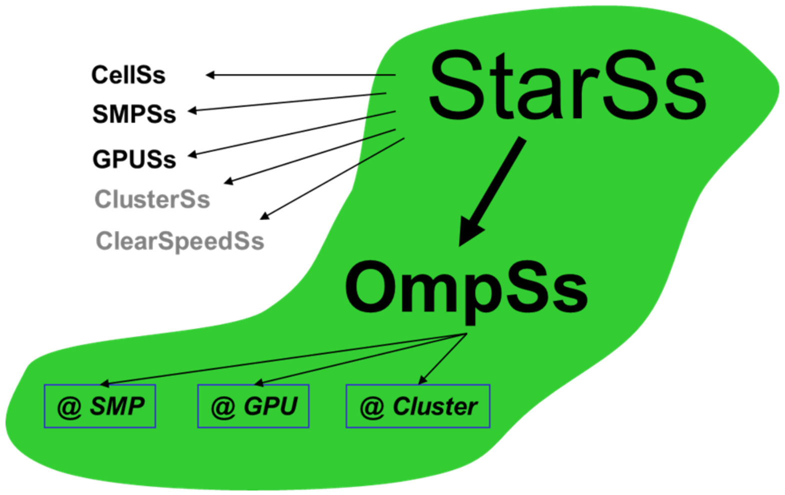
Is any of them more important than the others?

Which?

The sword to cut the “multicore” Gordian Knot

StarSs: a pragmatic approach

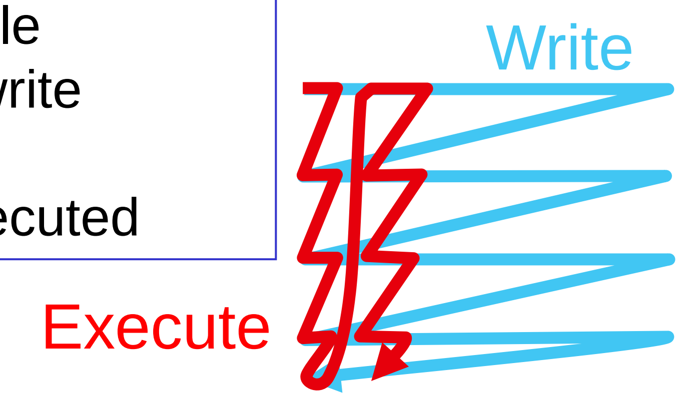
- Rationale
  - Runtime managed, asynchronous data-flow execution models are key
  - Need to provide a natural migration towards dataflow
  - Need to tolerate “acceptable” relaxation of pure models
  - Focus on algorithmic structure and not so much on resources

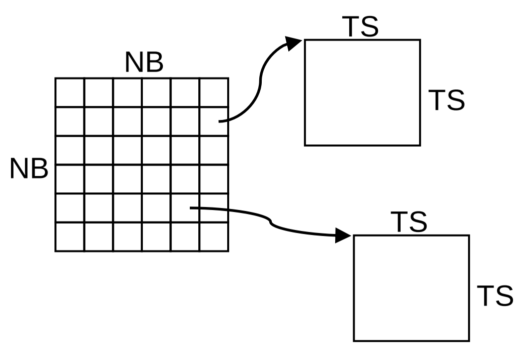


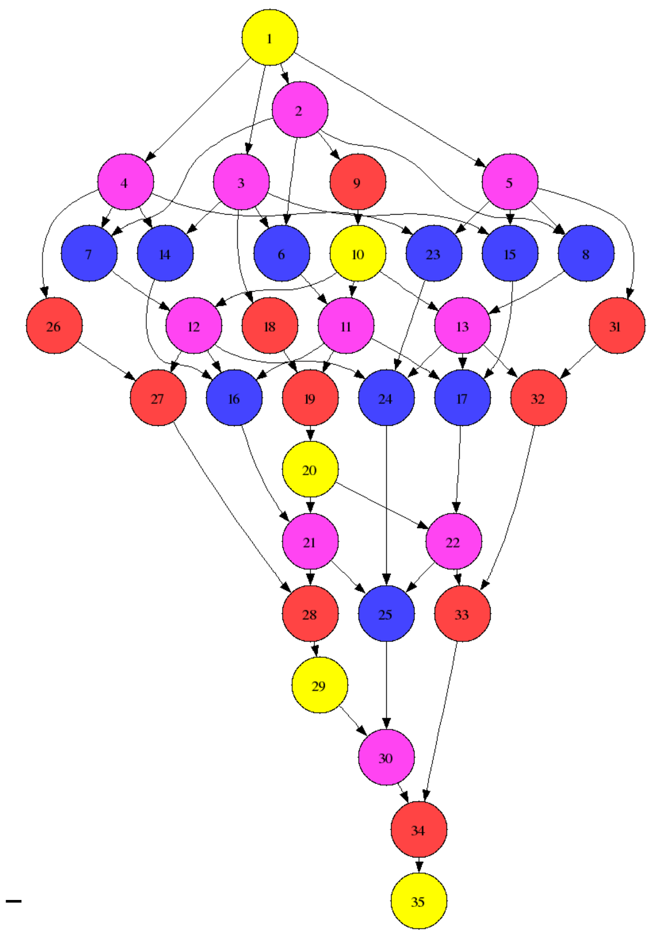
- StarSs: a family of **task based** programming models
  - Basic concept: **write sequential on a flat single address space** + **directionality annotations**
    - Order **IS** defined !!!
    - Dependence and data access related **information** (NOT specification) in a single mechanism
    - Think global, specify local
    - Power to the runtime !!!

StarSs: data-flow execution of sequential programs

Decouple
how we write
form
how it is executed





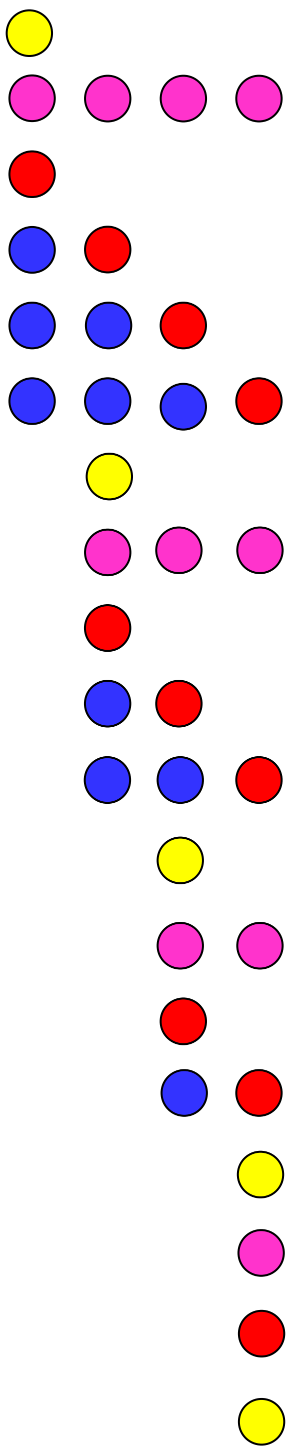


```
void Cholesky( float *A ) {
   int i, j, k;
   for (k=0; k<NT; k++) {
      spotrf (A[k*NT+k]) ;
      for (i=k+1; i<NT; i++)
         strsm (A[k*NT+k], A[k*NT+i]);
      // update trailing submatrix
      for (i=k+1; i<NT; i++) {
         for (j=k+1; j<i; j++)
            sgemm( A[k*NT+i], A[k*NT+j],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


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

StarSs vs OpenMP



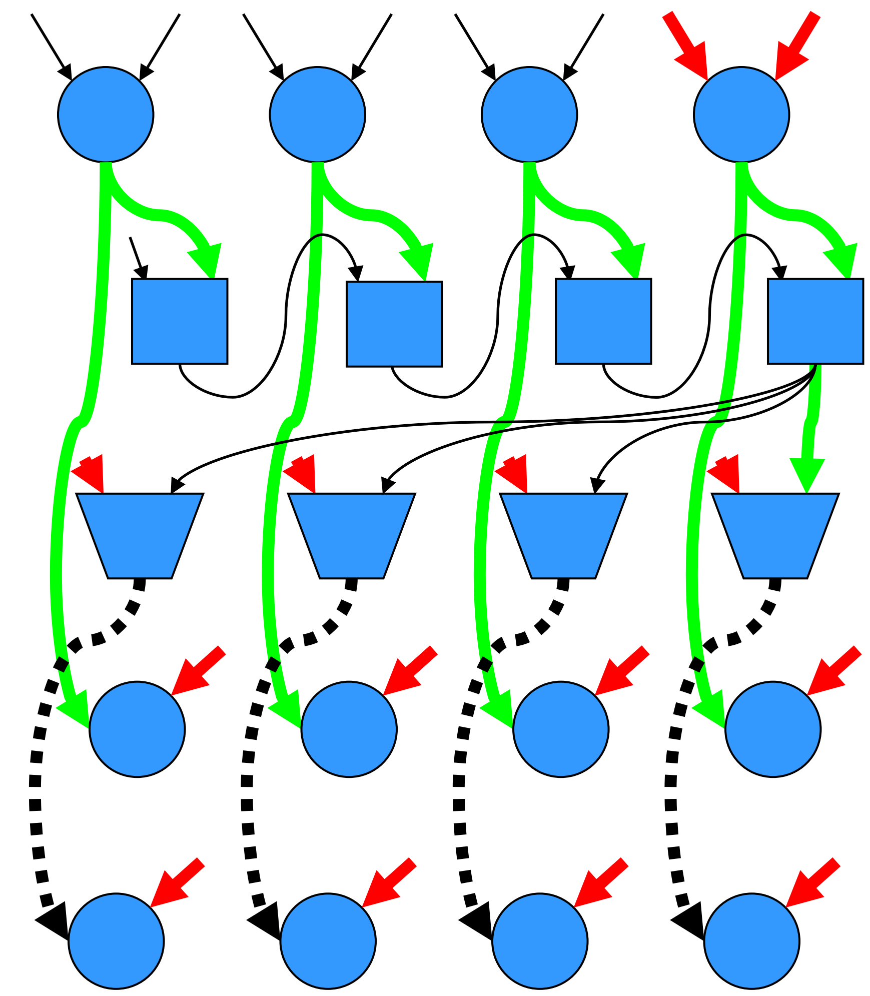
```
void Cholesky( float *A ) {
   int i, j, k;
   for (k=0; k<NT; k++) {
      spotrf (A[k*NT+k]);
      #pragma omp parallel for
      for (i=k+1; i<NT; i++)
         strsm (A[k*NT+k], A[k*NT+i]);
      for (i=k+1; i<NT; i++) {
         #pragma omp parallel for
         for (j=k+1; j<i; j++)
            sgemm( A[k*NT+i], A[k*NT+j], A[j*NT+i]);
         ssyrk (A[k*NT+i], A[i*NT+i]);
   }
}
```

```
         strsm (A[k*NT+k], A[k*NT+i]);
      for (i=k+1; i<NT; i++) {
         for (j=k+1; j<i; j++) {
            #pragma omp task
            sgemm( A[k*NT+i], A[k*NT+j], A[j*NT+i]);
         }
      #pragma omp task
      ssyrk (A[k*NT+i], A[i*NT+i]);
      #pragma omp taskwait
   }
```

```
void Cholesky( float *A ) {
   int i, j, k;
   for (k=0; k<NT; k++) {
      spotrf (A[k*NT+k]);
      #pragma omp parallel for
      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++) {
         #pragma omp task
         {
         #pragma omp parallel for
         for (j=k+1; j<i; j++)
            sgemm( A[k*NT+i], A[k*NT+j], A[j*NT+i]);
         }
         #pragma omp task
         ssyrk (A[k*NT+i], A[i*NT+i]);
         }
         #pragma omp taskwait
   }
}
```

StarSs: the potential of data access information

- **Flat global address space seen by programmer**
- Flexibility to dynamically traverse dataflow graph "optimizing"
  - Concurrency. Critical path
  - Memory access: data transfers performed by run time
- Opportunities for runtime to
  - Prefetch
  - Reuse
  - Eliminate antidependences (rename)
  - Replication management
    - Coherency/consistency handled by the runtime

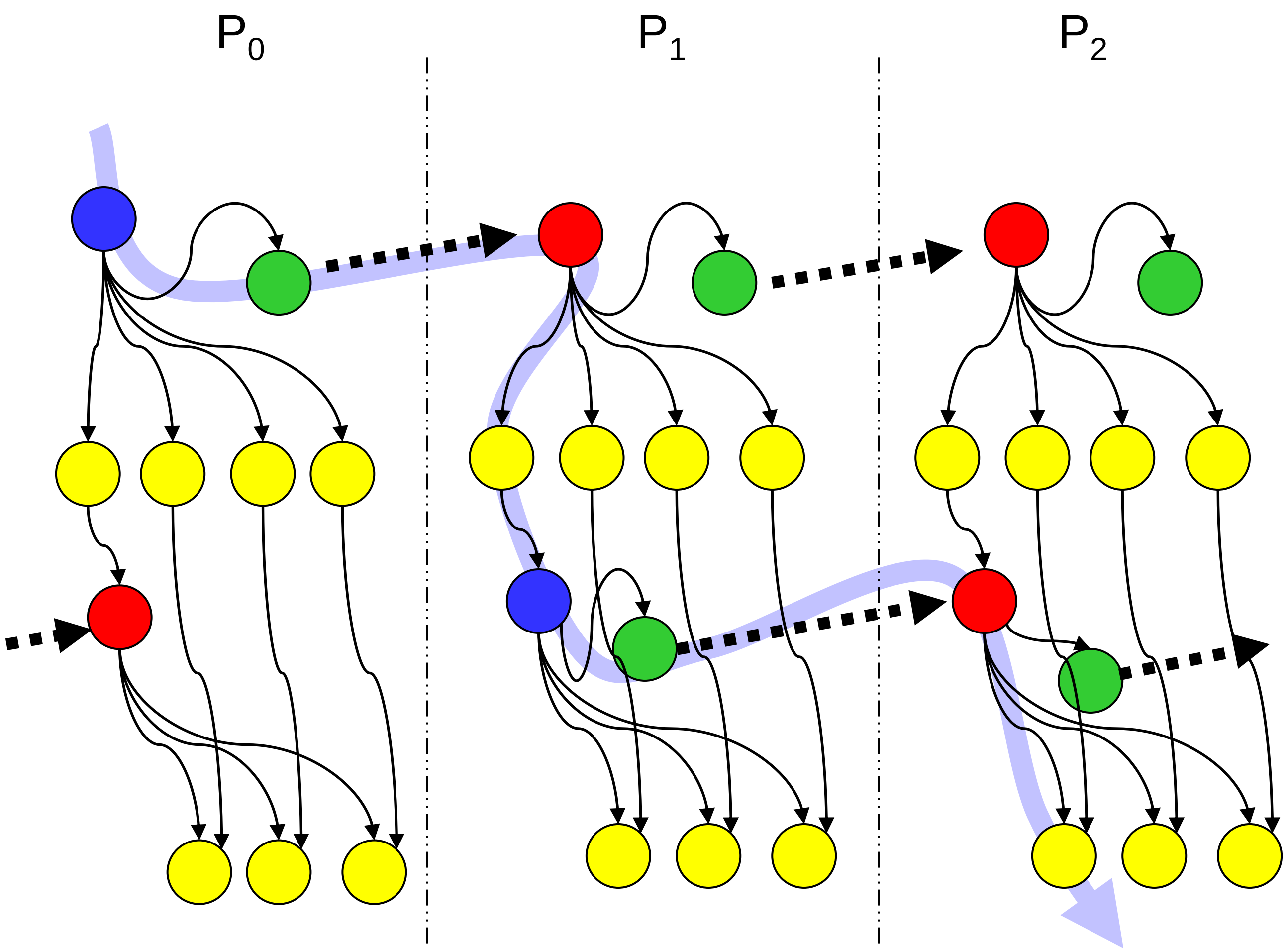


Hybrid MPI/StarSs

- Overlap communication/computation
- Extend asynchronous data-flow execution to outer level
- Linpack example: Automatic lookahead

```
…
for (k=0; k<N; k++) {
   if (mine) {
      Factor_panel(A[k]);
      send (A[k])
   } else {
      receive (A[k]);
      if (necessary) resend (A[k]);
   }
   for (j=k+1; j<N; j++)
      update (A[k], A[j]);
…
```

```
#pragma css task inout(A[SIZE])
void Factor_panel(float *A);
#pragma css task input(A[SIZE]) inout(B[SIZE])
void update(float *A, float *B);
```




```
#pragma css task input(A[SIZE])
void send(float *A);
#pragma css task output(A[SIZE])
void receive(float *A);
#pragma css task input(A[SIZE])
void resend(float *A);
```

V. Marjanovic, et al, “Overlapping Communication and Computation by using a Hybrid MPI/SMPSs Approach” ICS 2010

All that easy/wonderful?

- Difficulties for adoption
  - Chicken and egg issue users ↔ manufacturers
  - Availability.
    - Runtime implementations chasing new platforms
  - Development as we go
    - Fairly stable, minimal application update cost.
    - Happens to all models, by all developers (companies, research,…)
- Lack of program development support
  - Understand application dependences
  - Understand potential and best direction
- Difficulties of the models themselves
  - Simple concepts take time to be matured
  - As clean/elegant as we claim?
  - Legacy sequential code less structured than ideal

Early adopters and porting
Research support:
- **Consolider (Spain)**
- **ENCORE, TEXT, Montblanc, DEEP (EC)**

Standardization:
- OpenMP, …
- Maturity

New tools
- **Taskification**
- **Performance prediction**
- **Debugging**

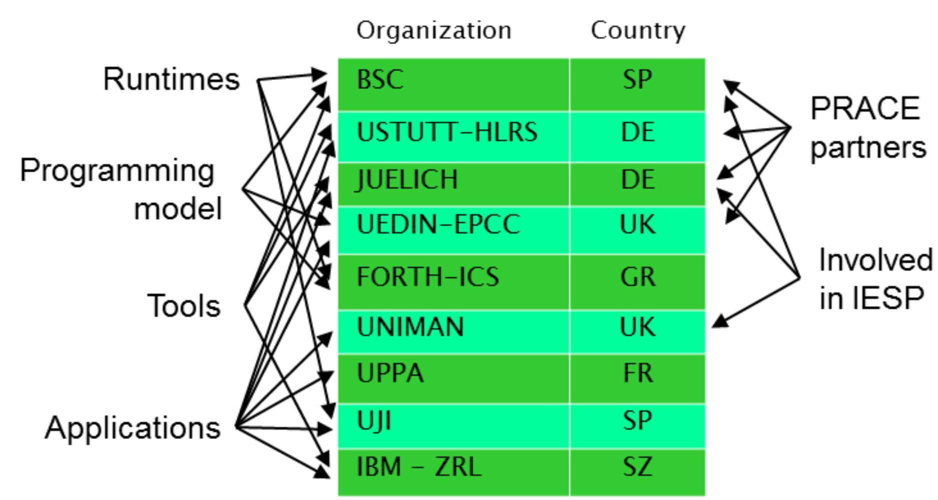
New Platforms
- **ARM + GPUs**
- **MIC**
- **…**

Examples
Training
Education

The TEXT project

- Towards EXaflop applicaTions (EC FP7 Grant 261580)

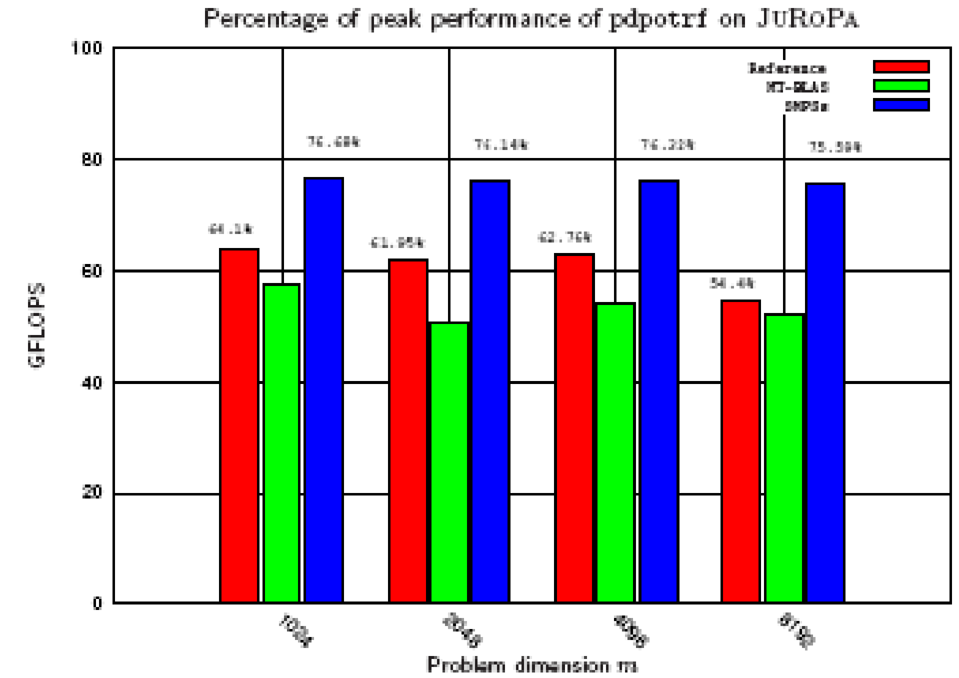
- Demonstrate that **Hybrid MPI/SMPSs** addresses the Exascale challenges in a an productive and efficient way.
  - Deploy at supercomputing centers: Julich, EPCC, HLRS, BSC
  - Port Applications (HLA, SPECFEM3D, PEPC, PSC, BEST, CPMD, LS1 MarDyn) and develop algorithms.
  - Develop additional environment capabilities
    - tools (debug, performance)
    - improvements in runtime systems (load balance and GPUSs)
  - Support other users
    - Identify users of TEXT applications
    - Identify and support interested application developers
  - Contribute to Standards (OpenMP ARB, PERI-XML)



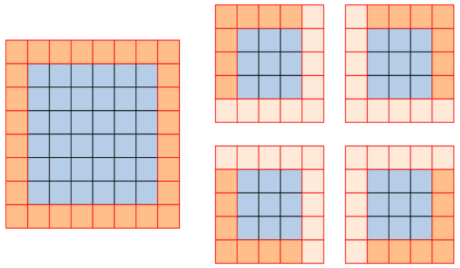
Deployment

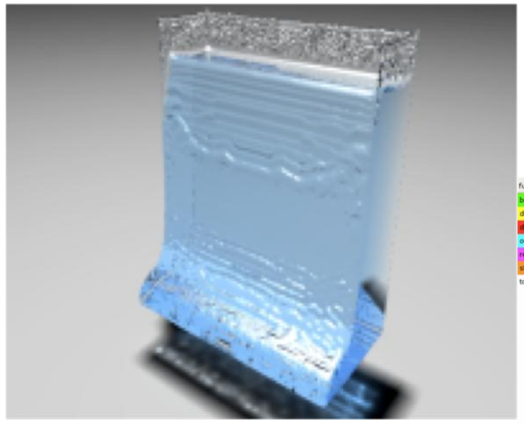
Name / Institution	Type of processor	Total core count
MareNostrum (BSC)	Power PC 970 MP	10240
JuGene (JSC)	32-bit PowerPC 450	294912
JuRoPA (JSC)	Dual-Quadcore Intel Nehalem	26304
Laki NEC Nehalem cluster (HLRS)	Intel Xeon (X5560) Nehalem	5600
Cray XT5m (HLRS)	Quad-Core AMD Opteron	896
HECToR Cray XE6 (EPCC)	12-core AMD Opteron Magny Cours	44544
Hermit Cray XE6 (HLRS)	AMD Opteron 8-Core, 2GHz Magny Cours	1344

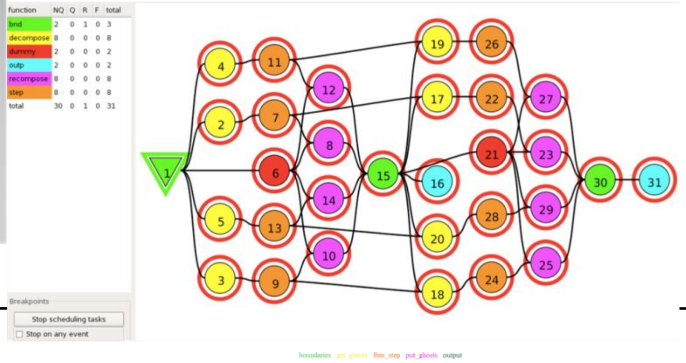
Codes being ported

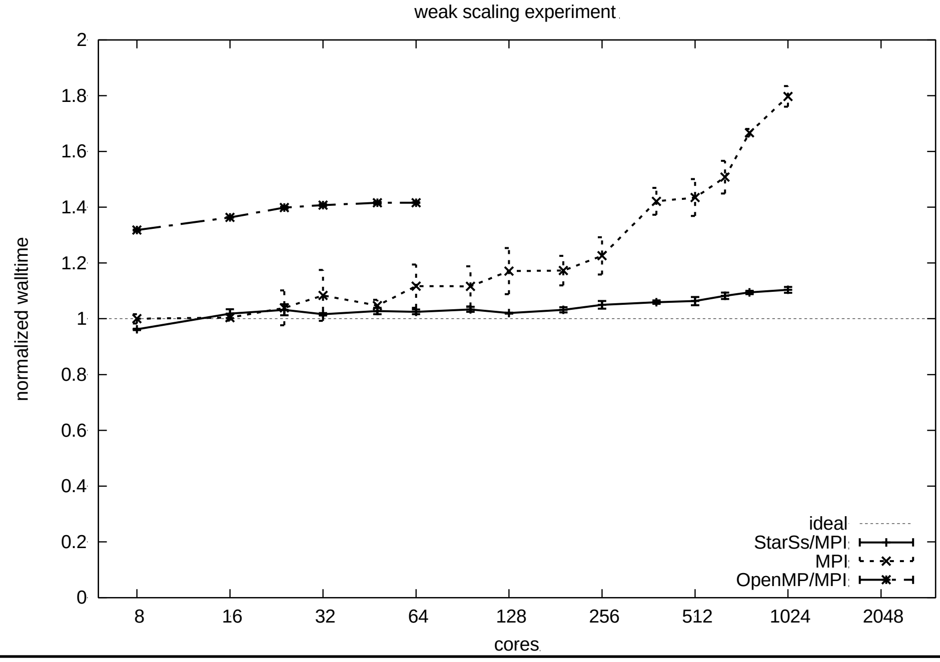


- Scalapack: Cholesky factorization (UJI)
  - Example of the issues in porting legacy code
  - Demonstration that it is feasible
  - The importance of scheduling
- LBC Boltzmann Equation Solver Tool (HLRS)
  - Solver for incompressible flows based on Lattice-Boltzmann methods (LBM)
  - LBM well suited for highly complex geometries. Simplified implementation: lbc
  - Stencil. Sub domains





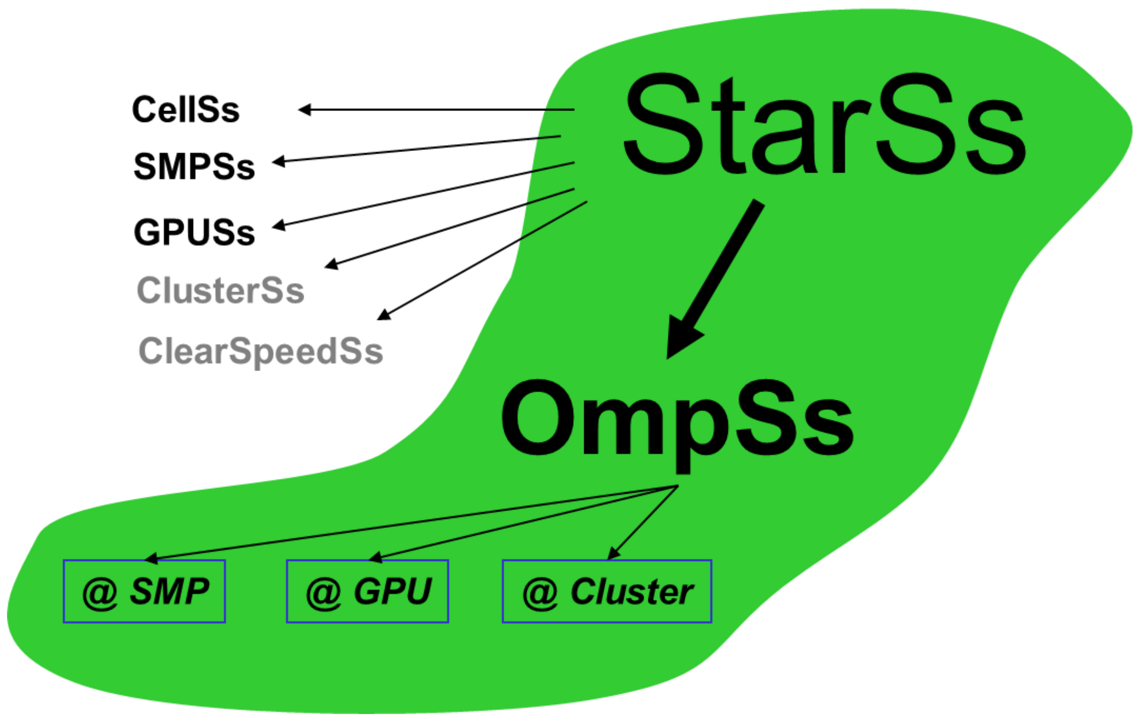




StarSs: history/strategy/versions

Basic SMPSs

- must provide directionality ∀argument
- Contiguous, non partially overlapped
- **Renaming**
- Several schedulers (priority, locality,…)
- No nesting
- C/Fortran
- **MPI/SMPSs optims.**



SMPSs regions

- C, No Fortran
- must provide directionality ∀argument
- **ovelaping &strided**
- **Reshaping strided accesses**
- **Priority and locality aware scheduling**

OMPSs

- C, C++, Fortran
- OpenMP compatibility (~)
- Contiguous and strided args.
- **Separate dependences/transfers**
- **Inlined/outlined pragmas**
- **Nesting**
- **Heterogeneity: SMP/GPU/Cluster**
- No renaming,
- **Several schedulers:** "Simple" locality aware sched,…

Evolving research since 2005

OmpSs

- What; Our long term infrastructure
  - “Acceptable” relaxation of basic StarSs concept
  - Reasonable merge/evolution of OpenMP

- Basic features
  - Inlined/outlined task specifications
    - Support multiple implementations for outlined tasks
  - Separation of information to compute dependences and data movement
    - Not necessary to specify directionality for an argument
  - Concurrent: Breaking inout chains (for reduction implementation)
  - Nesting
  - Heterogeneity: CUDA, OpenCL (in the pipe)
  - Strided and partially aliased arguments
  - C, C++ and Fortran

OmpSs: Directives

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

Support for multiple implementations of a task

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

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

```
#pragma omp task [ input (...)] [ output (...)] [ inout (...)] [ concurrent (...)]
 { function or code block }
```

To compute dependences

To allow concurrent execution of commutative tasks

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

Master wait for sons or specific data availability

Relax consistency to main program

CUDA support

```
#pragma omp target device(cuda)
__global__ void cuda_perlin (pixel output [], float time,
                             int j, int rowstride)
{
    unsigned int i = blockIdx.x * blockDim.x + threadIdx.x;
    unsigned int off = blockIdx.y * blockDim.y + threadIdx.y;
      float vdx = 0.03125f;
      float vdy = 0.0125f;
      float vs = 2.0f;
      float bias = 0.35f;
      float vx = 0.0f;
      float red, green, blue;
      float xx, yy;
      float vy, vt;

      vx = ((float) i) * vdx;
      vy = ((float) (j+off))
      vt = time * vs;

      xx = vx * vs;
      yy = vy * vs;

      red = noise3(xx, vt, yy
      green = noise3(vt, yy,
      blue = noise3(yy, xx, v

      red += bias;
      green += bias;
      blue += bias;

      // Clamp to within [0 .
      red = (red > 1.0f) ? 1.
      green = (green > 1.0f)
      blue = (blue > 1.0f) ?

      red = (red < 0.0f) ? 0.
      green = (green < 0.0f)
      blue = (blue < 0.0f) ?

      red *= 255.0f;
      green *= 255.0f;
      blue *= 255.0f;

      output[(off * rowstride
      output[(off * rowstride
      output[(off * rowstride
      output[(off * rowstride
}
```

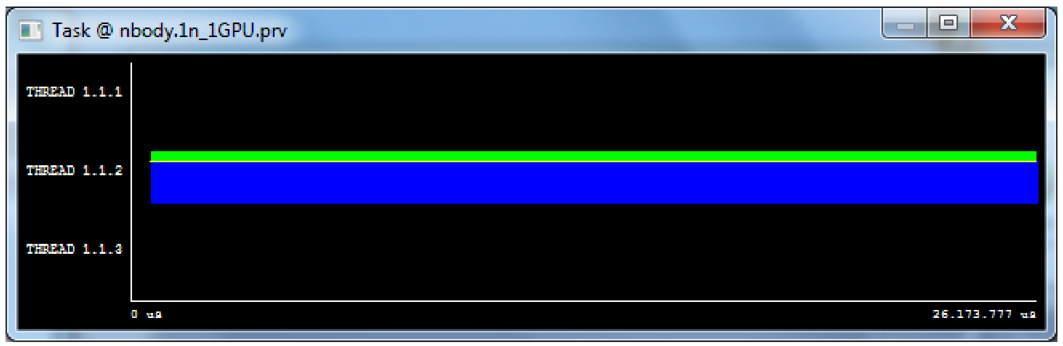
```
for (j = 0; j < img_height; j+=BS) {        // BS image rows per task
        pixel *out = &output[j*rowstride];
        #pragma omp target device(cuda) copy_deps
        #pragma omp task output([rowstride*BS]out)
        {
            dim3 dimBlock;
            dim3 dimGrid;
            dimBlock.x = (img_width < BSx) ? img_width : BSx;
            dimBlock.y = (BS < BSy) ? BS : BSy;
            dimBlock.z = 1;
            dimGrid.x = img_width/dimBlock.x;
            dimGrid.y = BS/dimBlock.y;
            dimGrid.z = 1;

            cuda_perlin <<<dimGrid, dimBlock>>> (out, time, j, rowstride);
        }
    }
#pragma omp taskwait noflush
```

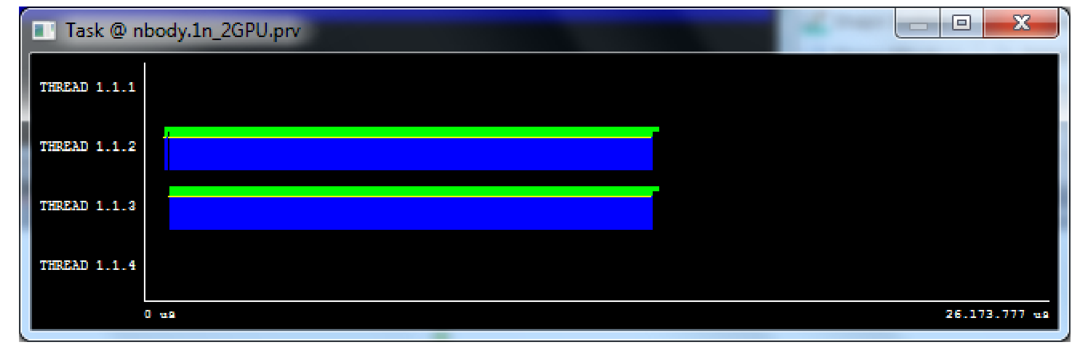
One source → many configurations of clusters with CUDA

1GPU

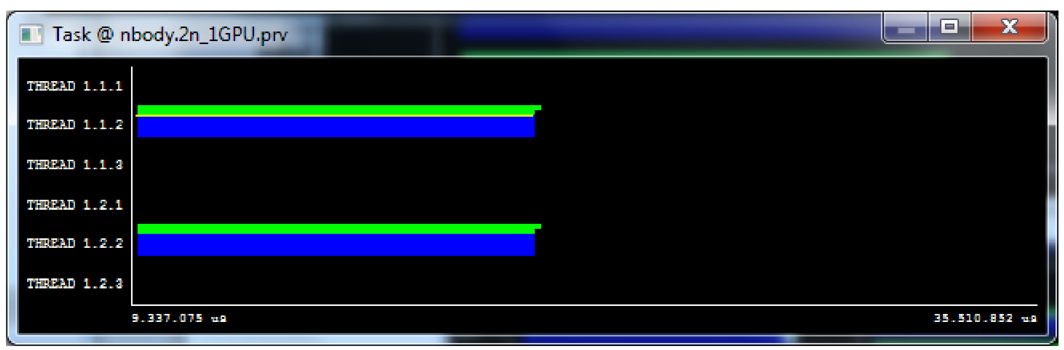
2 GPUs

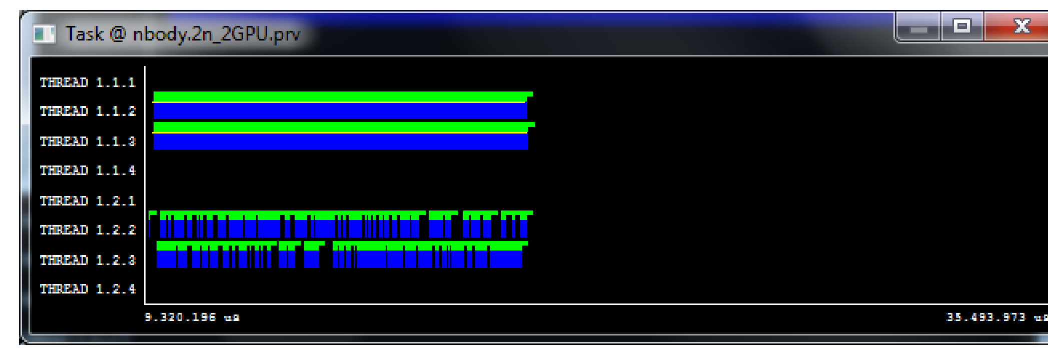
1 Node



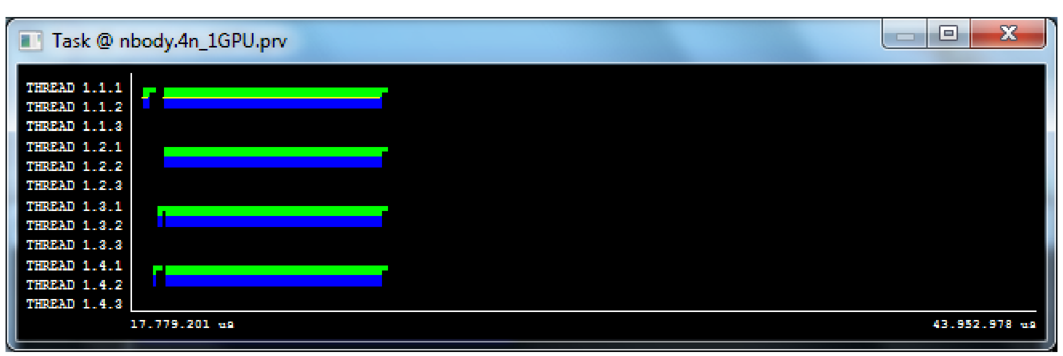


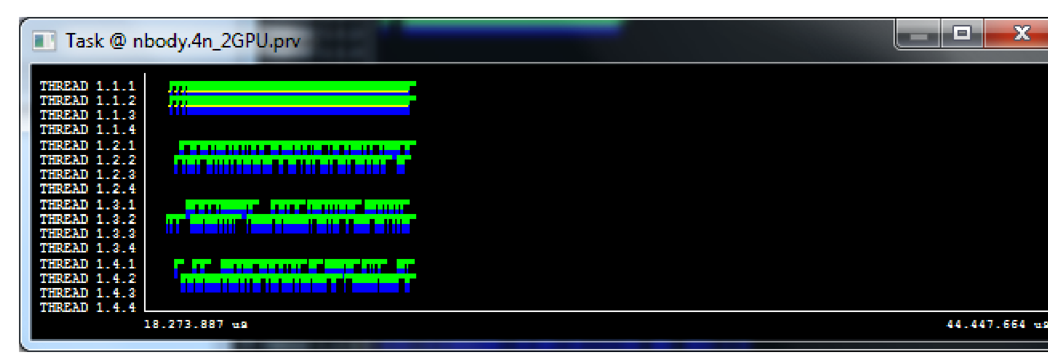
2 Nodes





4 Nodes





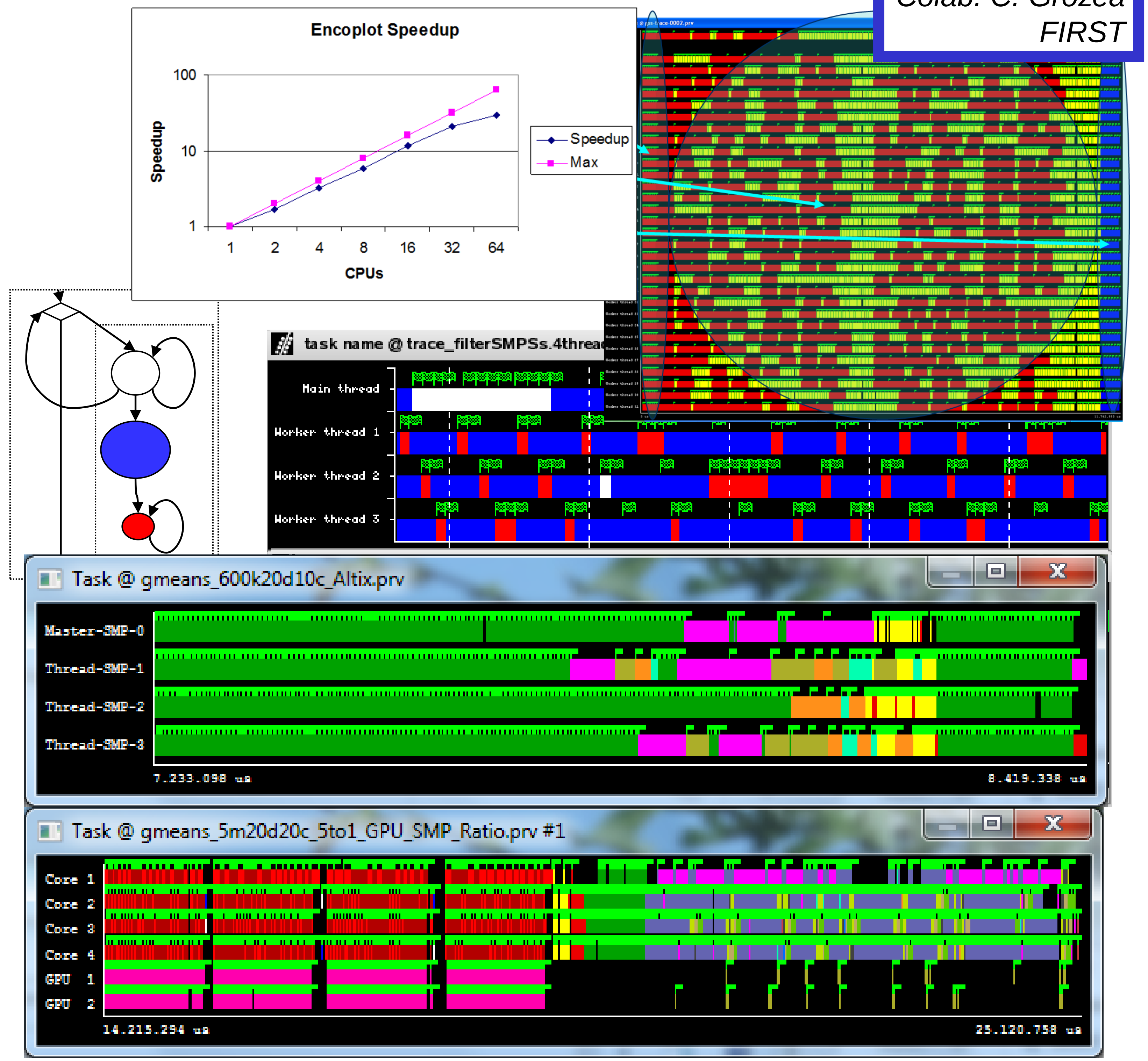
J. Bueno et al, “Productive Programming of GPU Clusters with OmpSs”, IPDPS2012

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

StarSs NOT only «scientific computing»

- Plagiarism detection
  - Histograms, sorting, …
- Trace browsing
  - Paraver
- Clustering algorithms
  - G-means
- Image processing
  - Tracking
- Embedded and consumer

Colab. C. Grozea
FIRST



encore

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Limitations?

- Discrete/atomic task
  - Run to completion task. Start and end only interaction points. No dependencies in/out to/from inside a task
  - Interactions half way through a task?
- Late dependence binding
  - Dependences are computed at task instantiation time.
  - Do we need mechanisms for later dependence computation?
- OmpSs relaxation of functional model
  - No need to specify directionality for all arguments, Commutative clause,…
  - Flexibility – risk tradeoff?

Limitations?

- Limitation in data access patterns
  - Contiguous/Strided regions
  - Need/can afford further structures? Irregularly scattered, pointer traversal, nested, …
- Granularity: flexibility vs. cost
  - Parallelism and lookahead more important than overhead

J.M. Perez et al, "Handling task dependencies under strided and aliased references" ICS 2010

  - When: determined at instantiation time: may be too early if too much lookahead
- How much of a limitation, alternatives, worthwhile? needed usage feedback

StarSs: Enabler for exascale

- Can exploit very unstructured parallelism
  - Not just loop/data parallelism
  - Easy to change structure
- Supports large amounts of lookahead
  - Not stalling for dependence satisfaction
- Allow for locality optimizations to tolerate latency
  - Overlap data transfers, prefetch
  - Reuse
- Nicely hybridizes into MPI/StarSs
  - Propagates to large scale the node level dataflow characteristics
  - Overlap communication and computation
  - A chance against Amdahl's law
- Homogenized view at heterogeneity
  - Any # and combination of CPUs, GPUs
  - Support autotuning
- Malleability: Decouple program from resources
  - Allowing dynamic resource allocation and load balance
  - Tolerate noise

Data-flow; Asynchrony

Potential is there;
Can blame runtime

Compatible with proprietary
low level technologies

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

A quiet revolution

- A change in mentality

Bottom up and being in total control Fork join, data parallel, explicit data placement	→	Top down, potentials and hints rather than how-tos, Asynchrony, data flow, automatic locality management

- Deeply rooted (in or genes), but need to overcome our fears.
  - May require some effort, but it is possible and there is a lot to gain.
  - Understanding and confidence through tools will be key
  - Need education from very early levels (shape instead of reshape minds)
- Adaptability/Flexibility is key to survive in rapidly changing environments