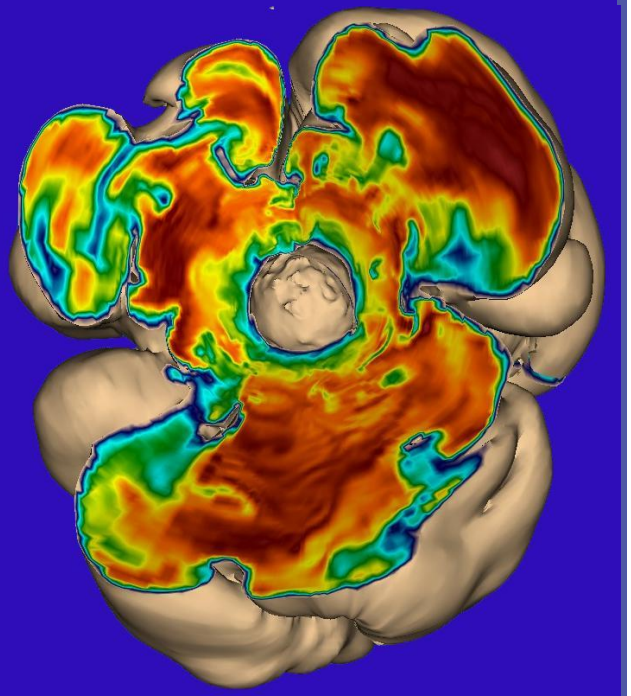


The challenges of massively parallel computations and the petaflop systems: the case of Supernova Research in Astrophysics

IDRIS, March 3rd

Andreas Marek

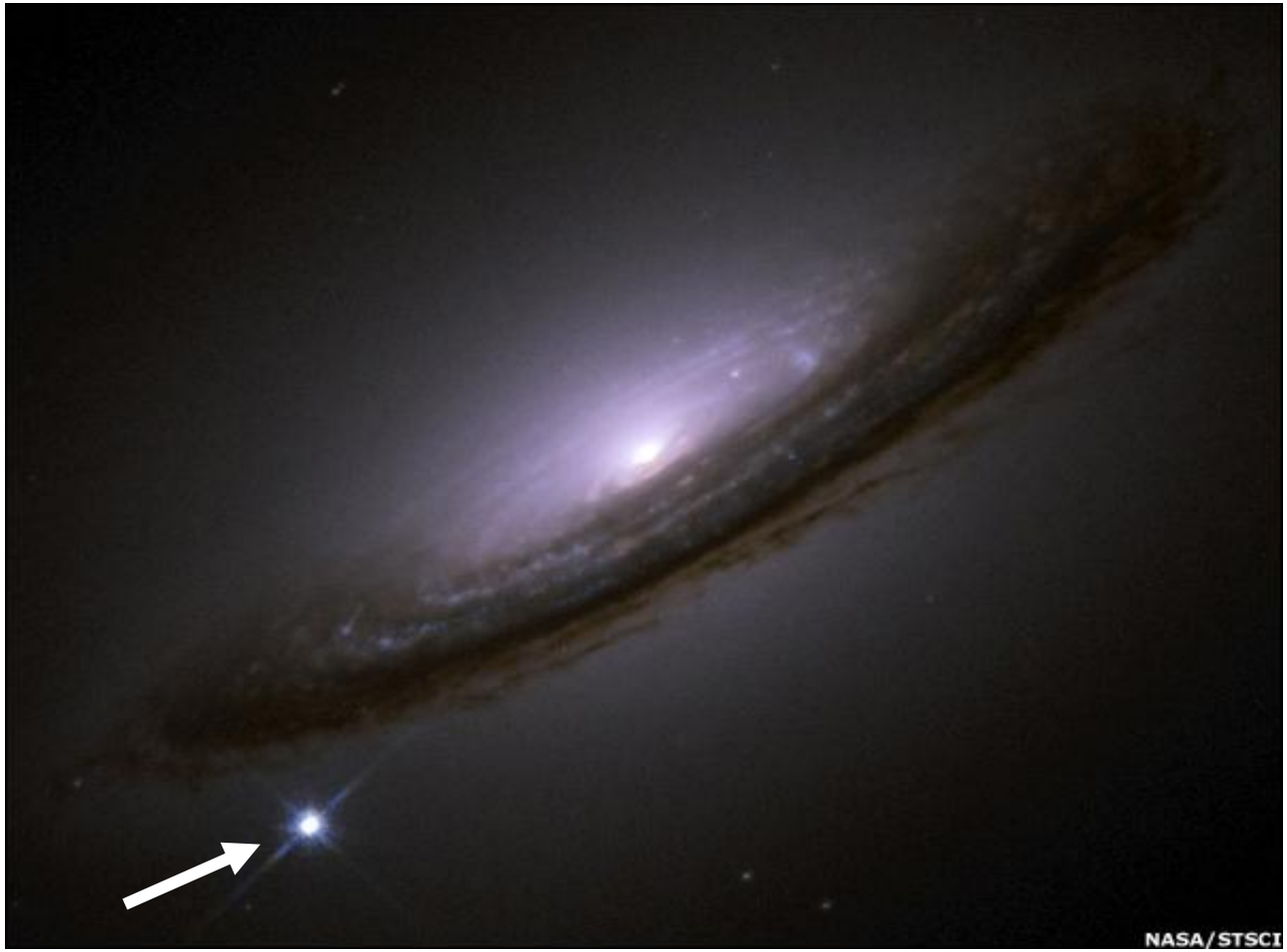
Computer Centre of Max-Planck Society

RZG
Rechenzentrum - Garching

Content

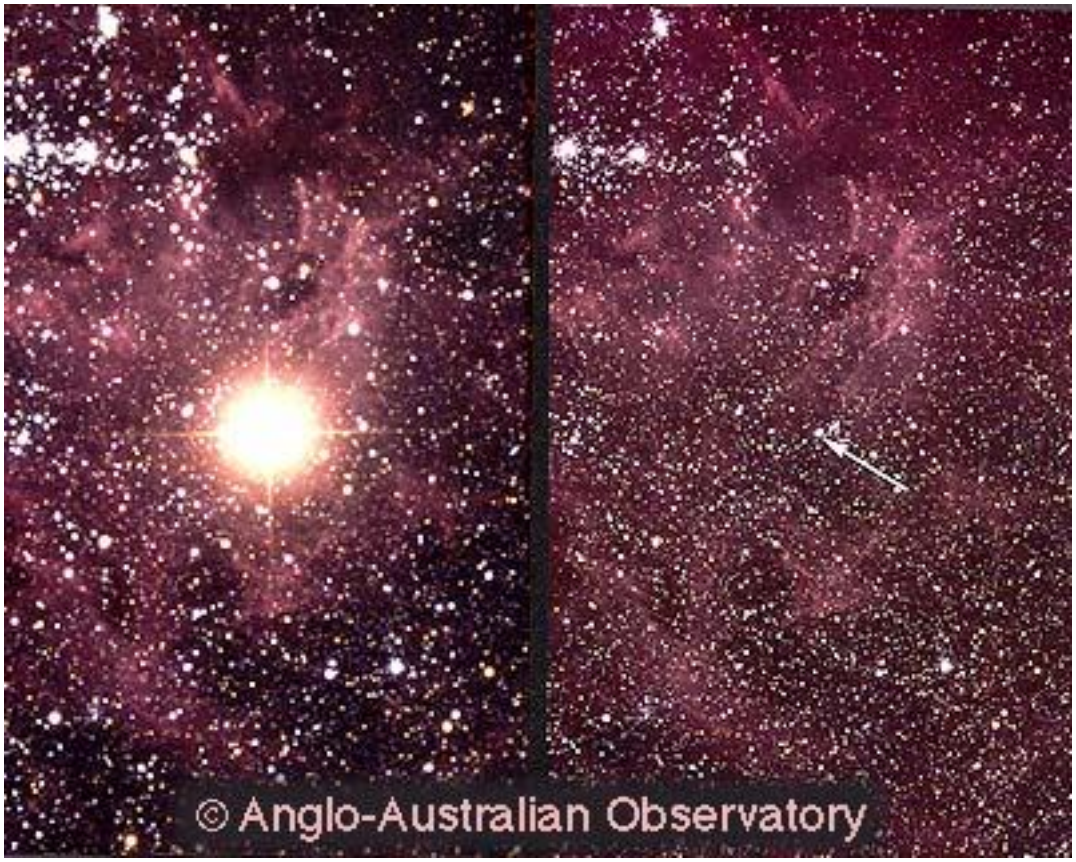
- **What is a supernova?**
  - **Observational facts**
  - **A simple model**
  - **Why are they interesting**
- **The VERTEX code in a nutshell**
  - **Algorithmic approach**
  - **Parallelization**
- **Porting to Bluegene/P**
  - **Determining the used memory**
  - **Handling of look-up tables**
  - **The usage of pointers**
  - **IO: parallel HDF5**
  - **Debuging on Bluegene**
  - **Scaling of VERTEX**
  - **MPI communications**
- **Summary**

What is a supernova? (Observations I)



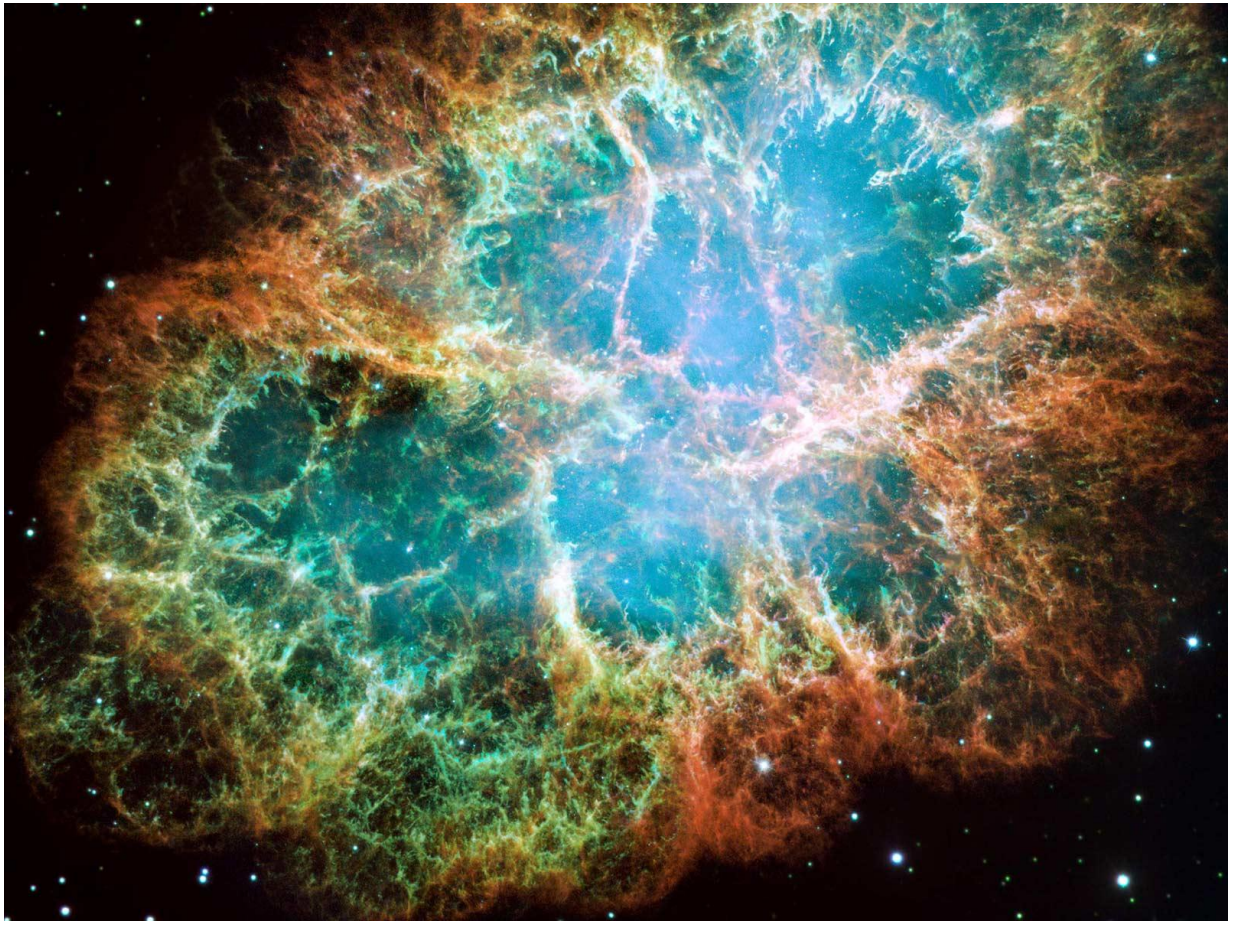
A single supernova does outshine an entire galaxy with its ~10^{10} stars

What is a supernova? (Observations II)



A single supernova does outshine an entire galaxy with its ~10^{10} stars, and it appears at the position of an ordinary star

What is a supernova? (Observations III)



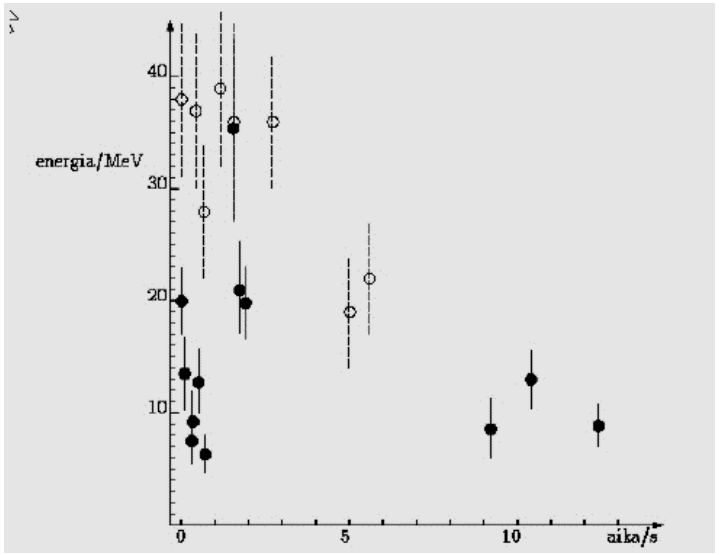
and leaves behind expanding remnants

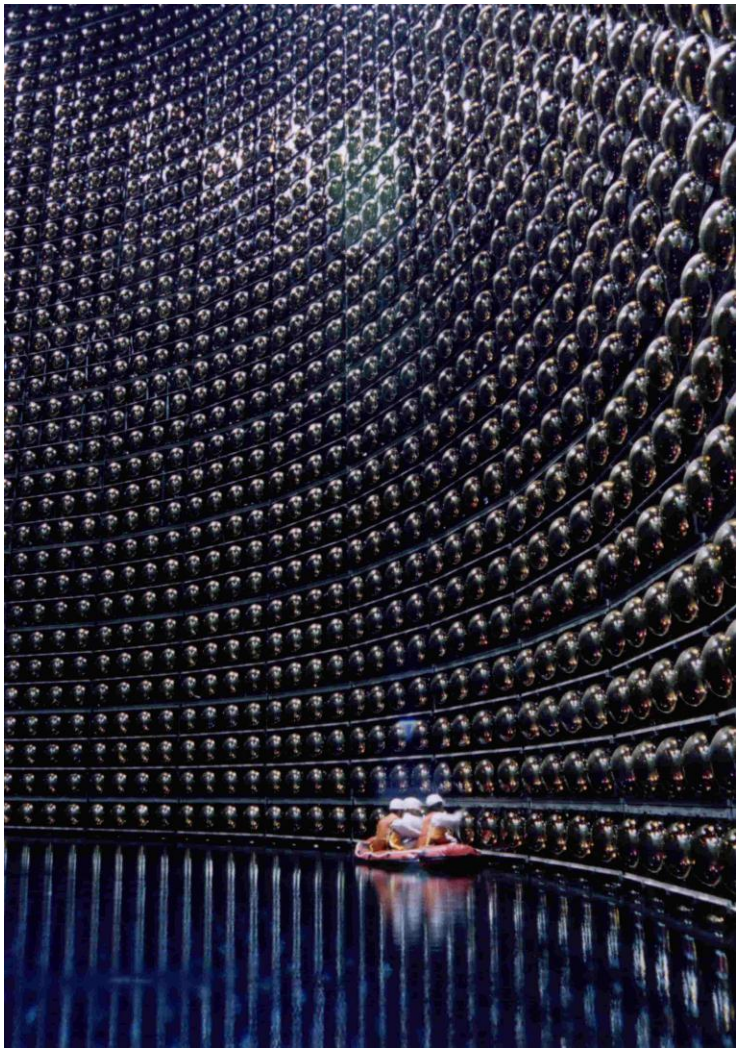
What is a supernova? (Observations IV)

SN 1987a in Large Magellanic Cloud (~10^5 Lyr):

19 neutrinos in IMB, Baksan, and Kamikokande II were detected

=> Principle idea of Core Collapse supernovae confirmed





Some observational facts

- Only ~ 10% of galactic supernovae are visible in light (dust)
- Extra-galactic SNe can be observed (due to extrem brightness)
- Often a remnant is left behind and pulsars (-> neutron stars) can be found

Masses and radii: $M_{star} > 8\ M_{solar}$, $R_{star} \sim 10^9$ km
$R_{IronCore} \sim 5000$ km
$M_{NeutronStar} \sim 1.4\ M_{solar}$, $R_{NeutronStar} \sim 10$ km

Energy budget: Released gravitational binding energy: $\sim 10^{53}$erg
Kinetic explosion energy: $\sim 10^{51}$ erg
Energy emitted in photons: $\sim 10^{49}$ erg

The rest is in neutrinos!

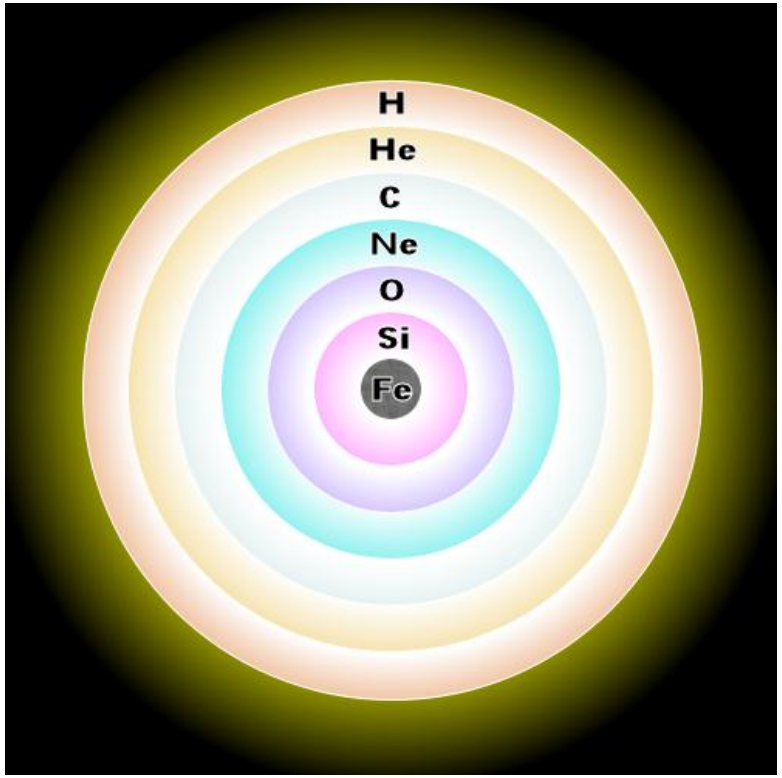
What is a supernova? – a simple model

A supernova does not mark the birth of a ``new super star´´ but rather the death of an old star:

During it`s life a star subsequently fusions (``burns´´) light elements (starting from hydrogen) to heavier onces.

This energy release produces enough pressure to counteract the pull of gravity of the self-gravitating plasma ball

Once, the inner centre has been burnded to iron, energy release ceases and the star dies in a supernova

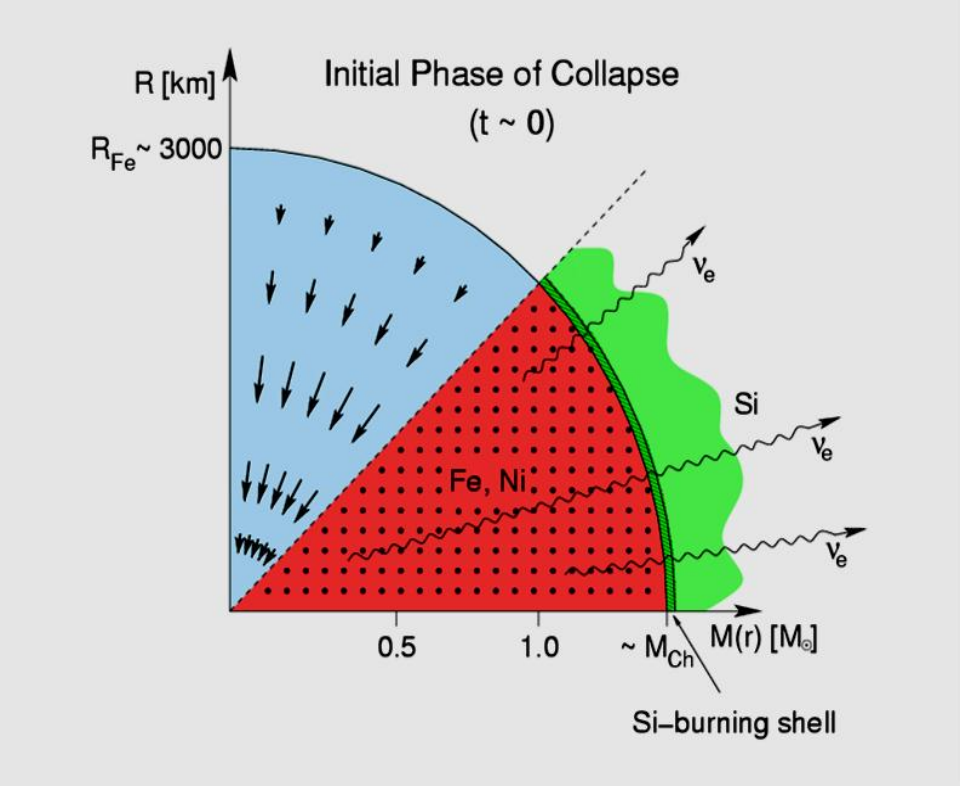


What is a supernova? – a simple model

A supernova does not mark the birth of a ``new super star´´ but rather the death of an old star:

The inner iron core begins to collapse, while the outer shells still continue to do nuclear burning

The collapse and thus rising density in the inner core cause enhanced neutrino emission

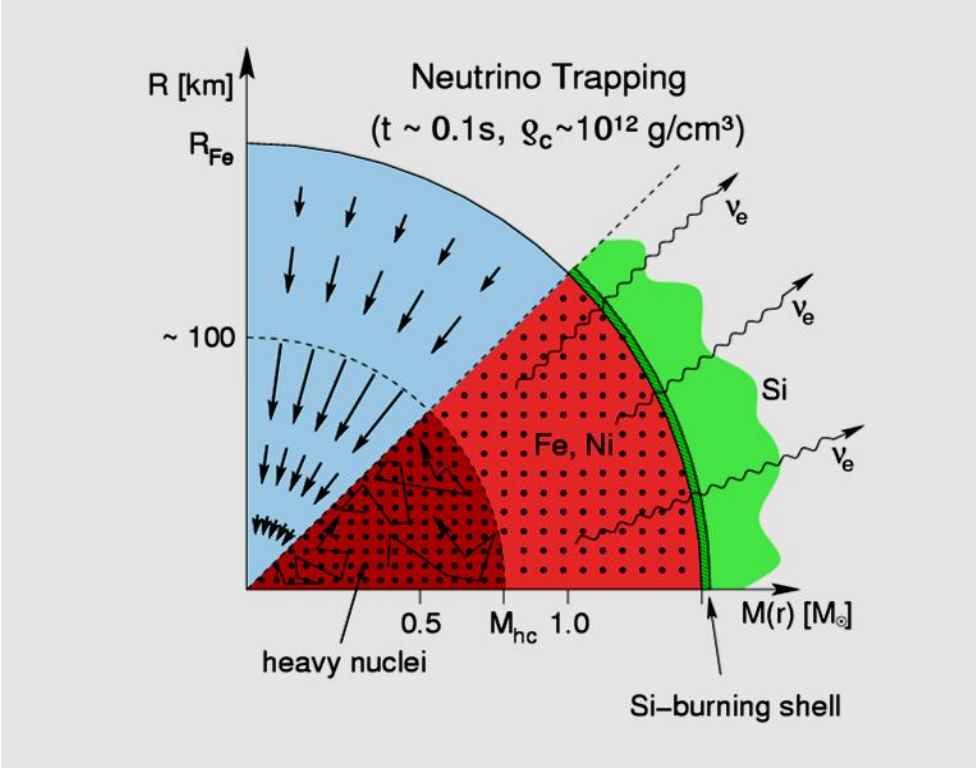


What is a supernova? – a simple model

A supernova does not mark the birth of a ``new super star´´ but rather the death of an old star:

When densities reach around 10^{12} g/ccm neutrinos become trapped in the collapsing material

At the same time the composition evolves from iron to heavier nuclei

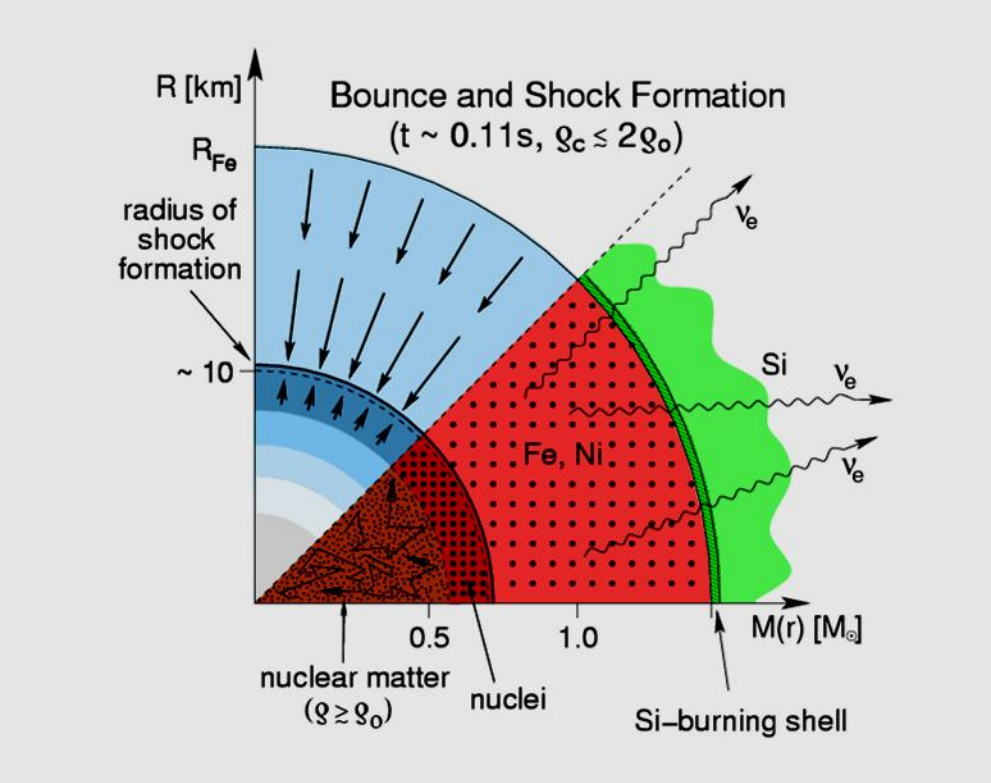


What is a supernova? – a simple model

A supernova does not mark the birth of a ``new super star´´ but rather the death of an old star:

When matter reaches nuclear matter density (~10^{14} g/ccm), a proto NS is formed, and the collapse stops abruptly (= bounce)

A shock wave forms and travels outwards through still collapsing material.

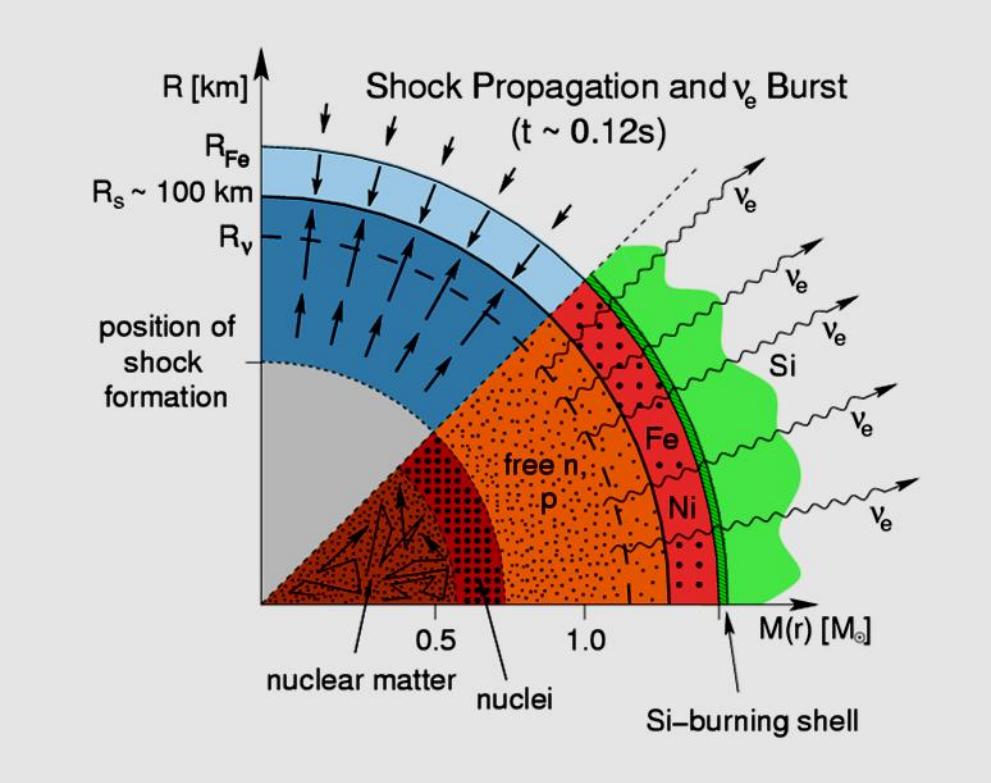


What is a supernova? – a simple model

A supernova does not mark the birth of a ``new super star´´ but rather the death of an old star:

A huge amount of neutrinos is released (neutrino burst), which could be detected

At the same time the shock wave loses it`s energy due to photo-disintegration of matter falling through the shock front

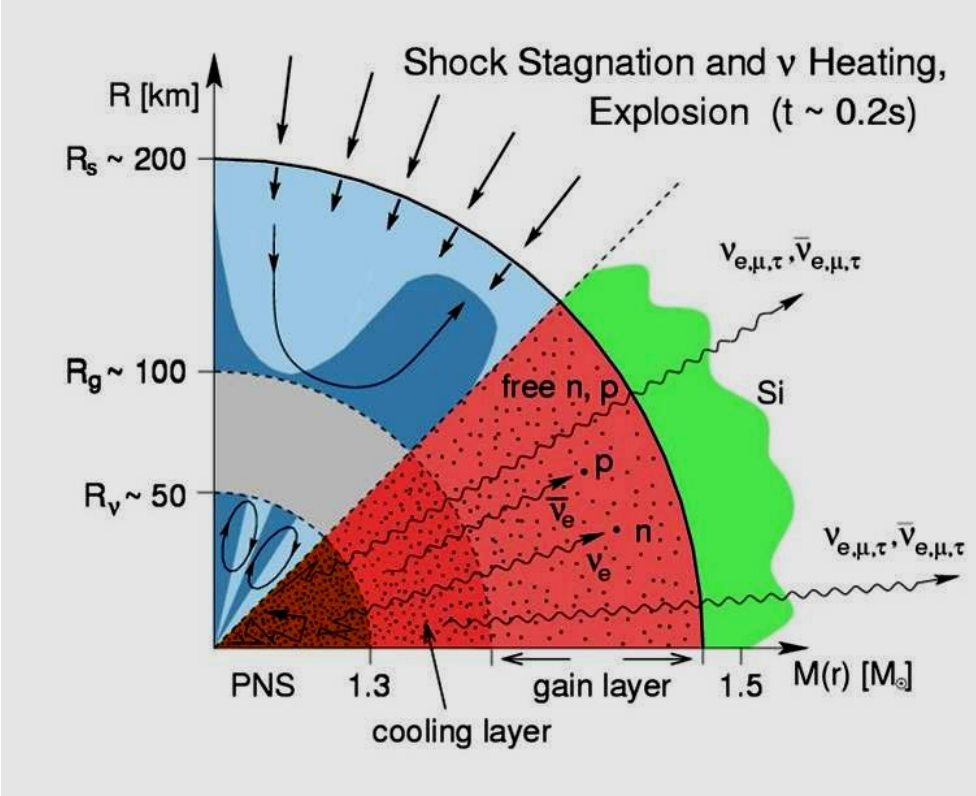


What is a supernova? – a simple model

A supernova does not mark the birth of a ``new super star´´ but rather the death of an old star:

On the time-scale of half a second, neutrinos emitted in the vicinity of the proto NS heat the matter and revive the shock expansion.

The development of instabilites and turbulent motion plays a crucial role

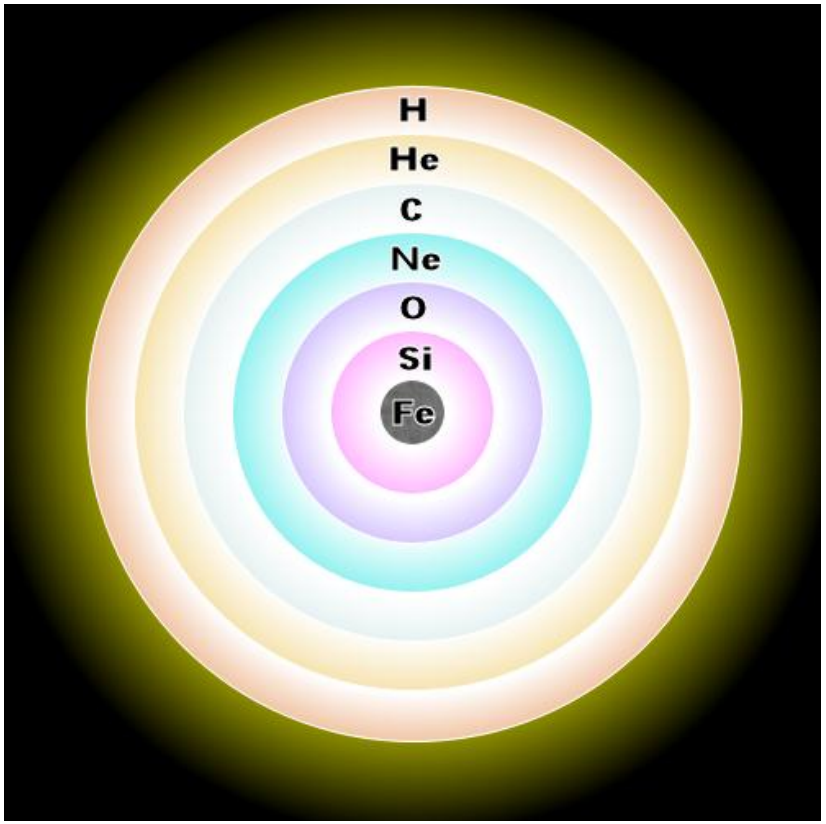


Why are supernovae interesting?

- Supernova explosions are among the most energetic events in the Universe (energy release $\sim 10^{53}$ erg)

- Supernova mix the heavy fusioned elements outwards and enrich the galactic and intergalactic medium with elements beyond H, and He

⇒every heavy element on Earth was processed in at least on star and expelled by a supernova !



The need for simulations

One cannot fly to a supernova, or do them in a laboratory (and it would not be healty anyway…)
Optical observations (at best): hours or days after the event started (-> scale problem), and only sparse carries of other information (e.g. neutrinos)

=> limited information ``black box´´ car analogy

a car where you cannot look under the hood

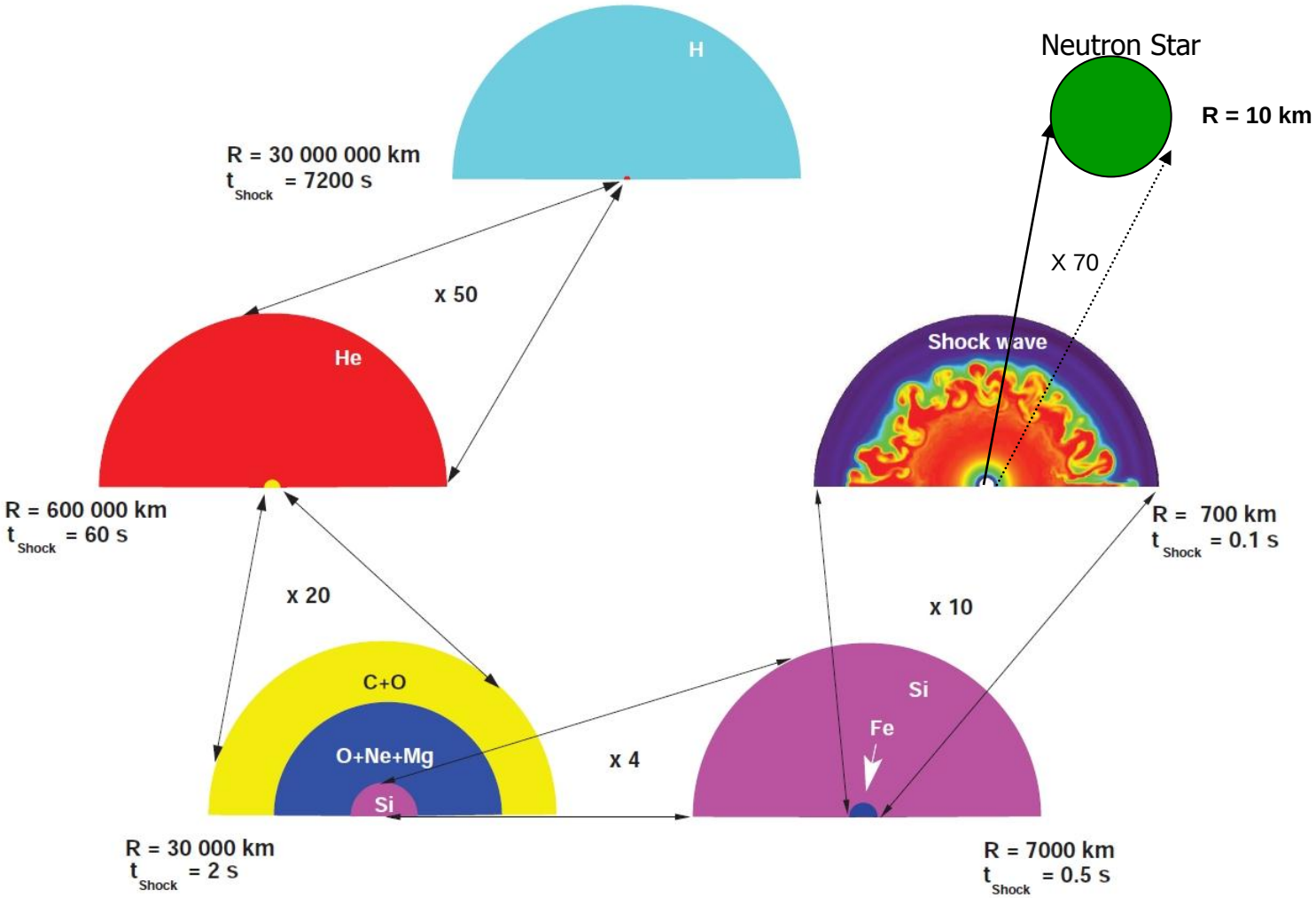
- you just know it needs fuel (SN: stellar material)
- you have to start it (SN: it starts somehow)
- the car gets warm, moves, expels some gas (SN: get an explosion)

But how does it work ?

⇒Take the car appart, solve problem by thinking and if it gets to complex, build a simulation and match with your observables

Simulations: The challenges I: scales

The scale problem: range at least 10^4 -> timestep limitations



The challenges II: physical complexity

Complexity Problem: in the last 4 decades it become clear that one has to include in the simulations

- **multi-dimensional hydrodynamics (stellar plasma, instabilities, turbulence)**
- **Gravity : general-relativity or at least reasonable approximation**
- **Nuclear physics:**
  - **High density matter (NS description)**
  - **Nuclear reaction network (Composition changes, energy source term)**
- **multi-dimensional spectral neutrino radiation transport (main sink of energy, responsible for driving explosion)**
  - **detailed kernels for neutrino-matter interactions**

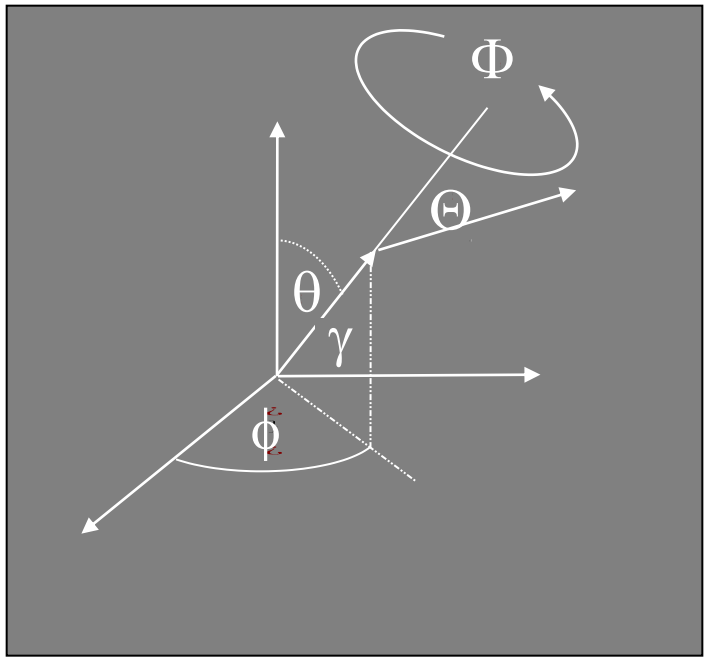
The challenges III: The curse of dimensions

- **Boltzmann equation determines neutrino distribution function in phase space**

$$f(r,\theta,\phi,\Theta,\Phi,\varepsilon,t)$$

- **Integration over momentum space yields source terms for hydrodynamics**

$$Q(r,\theta,\phi,t),\ dY_e(r,\theta,\phi,t)/dt$$



Solution approach

- **3D** hydro + **6D** direct discretization of Boltzmann Eq. (no serious attempt yet)
- **2D** hydro + **5D** direct discretization of Boltzmann Eq. (planned by DoE's TSI/SSC, abandoned)
- **2D or 3D** hydro + **"ray-by-ray-plus"** variable Eddington factor method (MPA, RZG)

Required resources

>= 10 PFlops (sustained!)

>= 10-100 TFlops/TBytes

>= 1TFlops – 1PFlops,
1 -100 TByte

"Ray-by-ray plus" variable Eddington factor method

- **Use angular moments to deal with integrodifferential character of Boltzmann eq.** $f(r,\theta,\phi,\Theta,\Phi,\varepsilon,t) \rightarrow J, H, K, L(r,\theta,\phi,\varepsilon,t)$

- **Use operator splitting:**

  - Consider lateral ν - advection and ν - pressure gradients (transp_advect_theta) $r_i: \partial/\partial t \ldots + \partial / \partial \theta \ldots = 0$
  - Do radial transport (transp_r) **"ray-by-ray"** $\theta_j: \partial / \partial t \ldots + \partial / \partial r \ldots + \partial / \partial \varepsilon \ldots = 0$

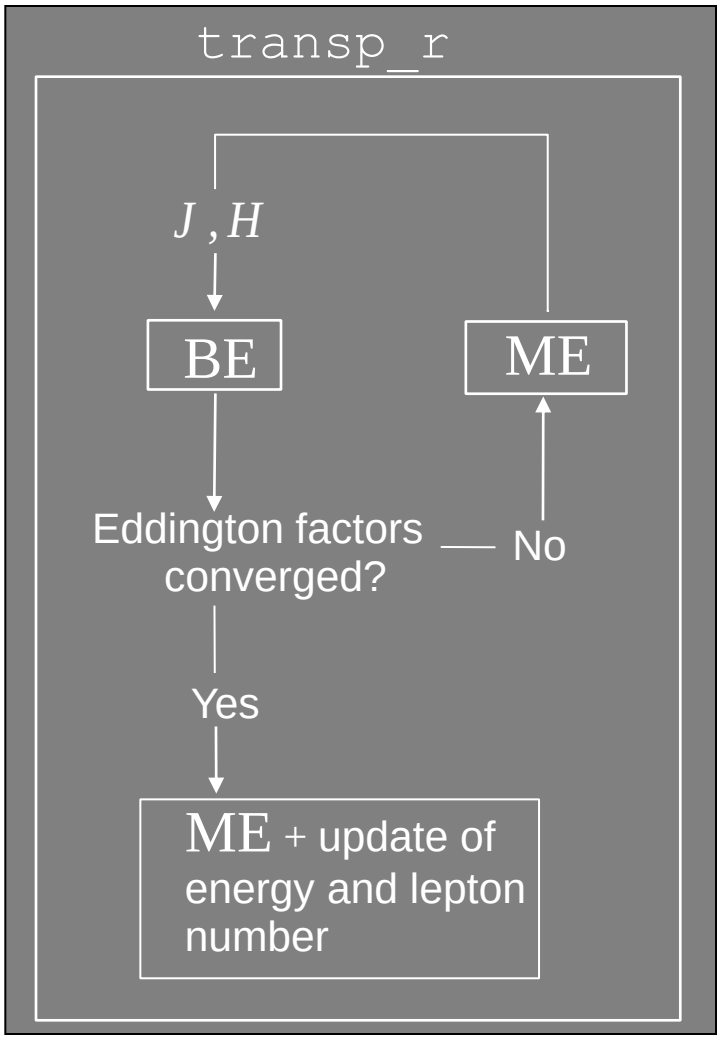
"Ray-by-ray plus" variable Eddington factor method (contd.)

- More moments (J, H, K, L) than moment eqs. (ME) of Boltzmann eq. (BE)

$$f_K = K/J \,,\; f_L = L/J$$

- closure by Eddington factors

- On each ray, obtain closure by iterative solution of moment equations, and a "model" Boltzmann equation whose rhs depends only on J, H



Moment Equations (ME)

- **To order** $O(\beta = v/c)$ **in comoving frame and on a radial ray**:

$$c^{-1} \partial U / \partial t + \beta \partial U / \partial t + r^{-2} \partial (r^2 F(U))/\partial r - \partial G(U)/\partial \varepsilon = S(U)$$

$$U = \begin{bmatrix} J \\ H \end{bmatrix}, F(U) = \begin{bmatrix} H \\ f_K J \end{bmatrix}, G(U) = \varepsilon \begin{bmatrix} \frac{\beta}{r}(1 - f_K) J + \frac{\partial \beta}{\partial r} + \frac{1}{c}\frac{\partial \beta}{\partial t} H \\ \frac{\beta}{r}(H - f_L) + \frac{\partial \beta}{\partial r} f_L J + \frac{1}{c}\frac{\partial \beta}{\partial t} f_K J \end{bmatrix}$$

- **System is hyperbolic for reasonable Eddington factors** $f_k = K/J$, $f_L = L/J$
- **Speed of light and stiff source term** $S(U)$ **severely restrict explicit time step!**

Implicit discretization

- Discretize MEs with backward time differencing (e.g. Euler, Gear, etc.):

$$U^{n+1}=U^{n}-\Delta t\, R(U^{n+1})$$

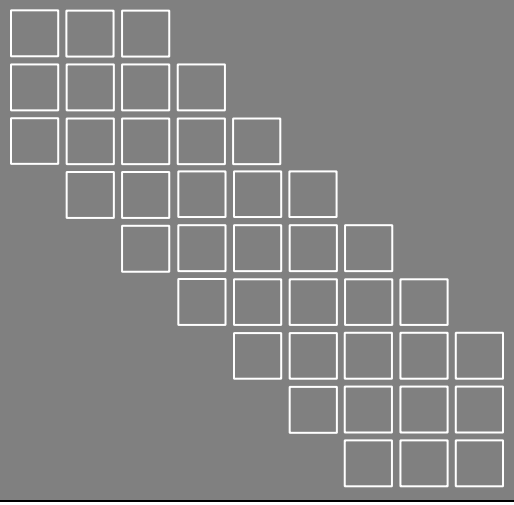
- Newton-Raphson:
  - Linearize the algebraic equations:
  - Solve the linear system for

$$[I+\Delta t\, \partial R/\partial U]\delta U = -\Delta t\, R(U^{n})$$

  - Iterate to convergence

$$\delta U = U^{n+1} - U^{n}$$

- Sructure of Jacobian for moment eqs. in PROMETHEUS/VERTEX supernova code:



Dense Blocks: Coupling in energy, due to source terms (collision term of Boltzmann eq.)

Diagonals: Coupling in radius

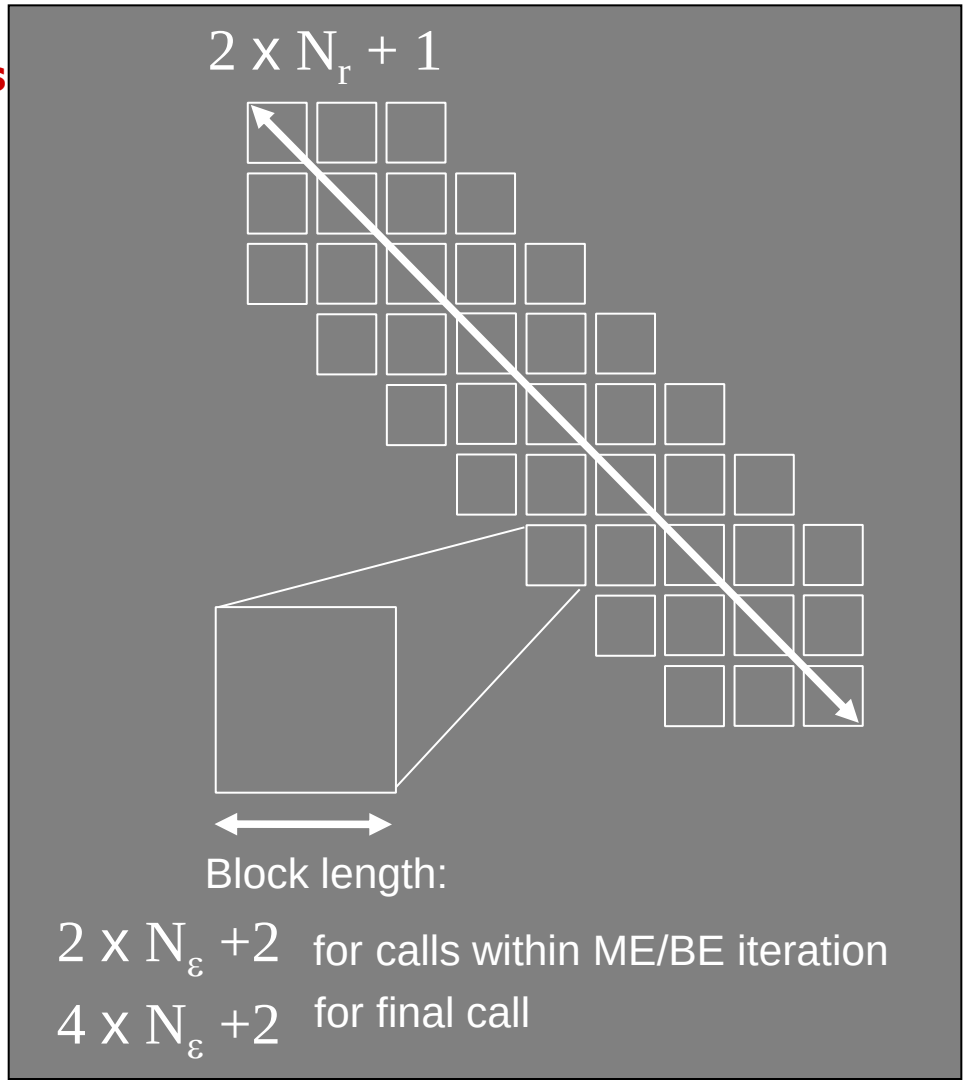
Inversion of block-pentadiagonal matrix

Size of matrix ~ 60000^2

=> in 3D : 20000 rays

corresponds to 20000 inversions

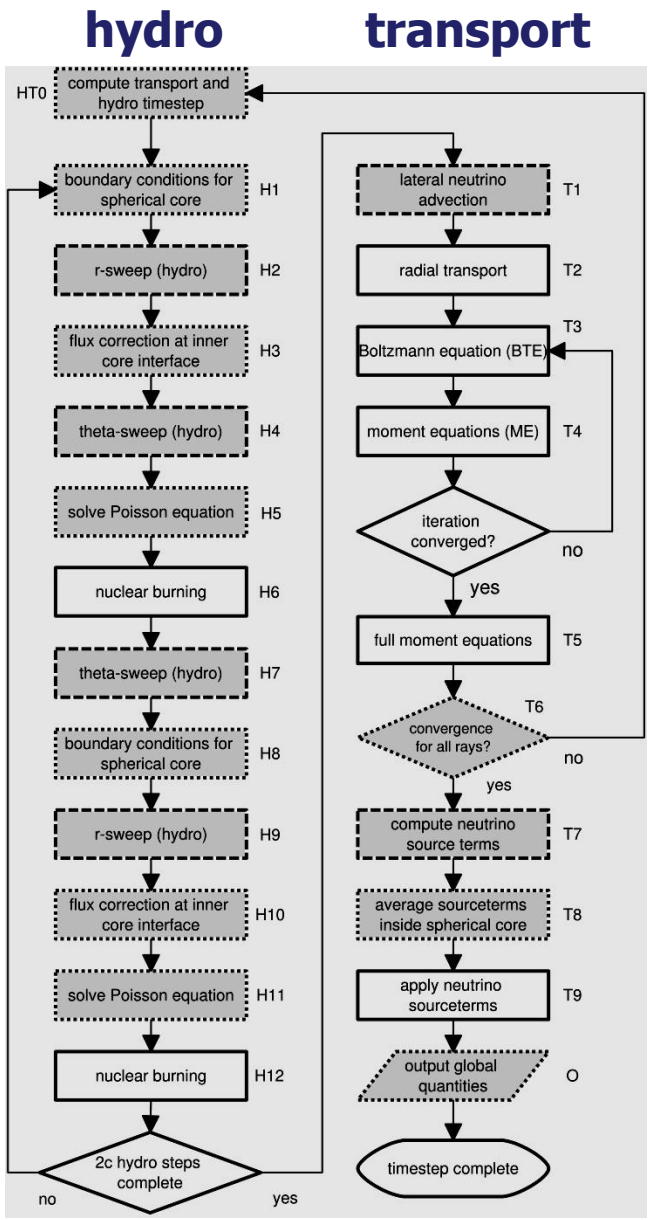
- **Among the major computational kernels**
- **Three direct solvers implemented at present**
  - **THOMAS: Block-Thomas algorithm vectorized over energy (i.e. within the blocks, using LAPACK, BLAS)**
  - **CYCLIC: Block-Cyclic-Reduction vectorized over energy**
  - **VCYCLIC: Block-Cyclic-Reduction vectorized over radius (i.e. along the diagonals)**



Flowchart of algorithm

next-neighbour communication

reduction operation

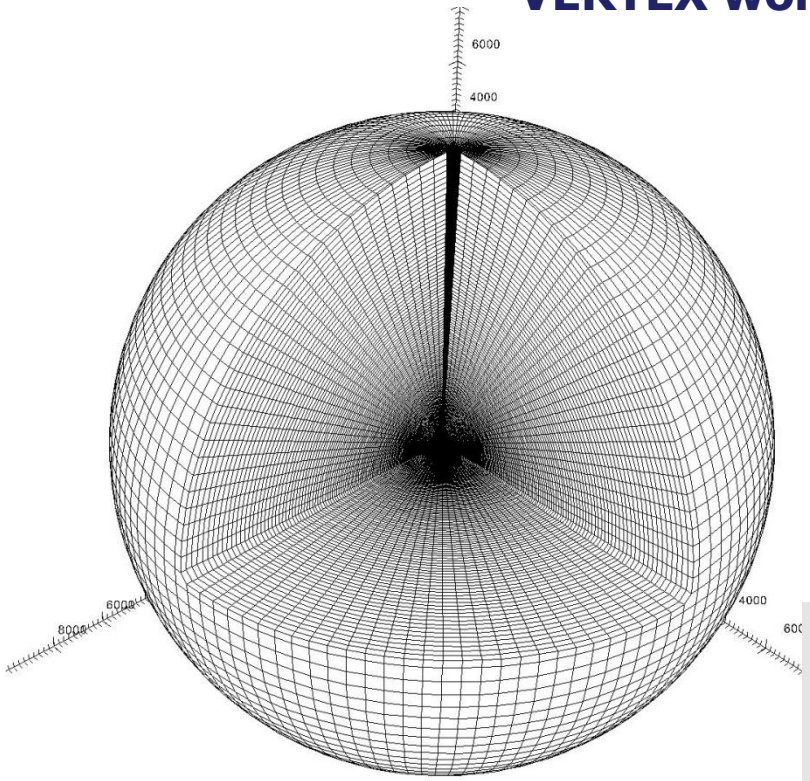


transport is almost communication free

The VERTEX code : Setup

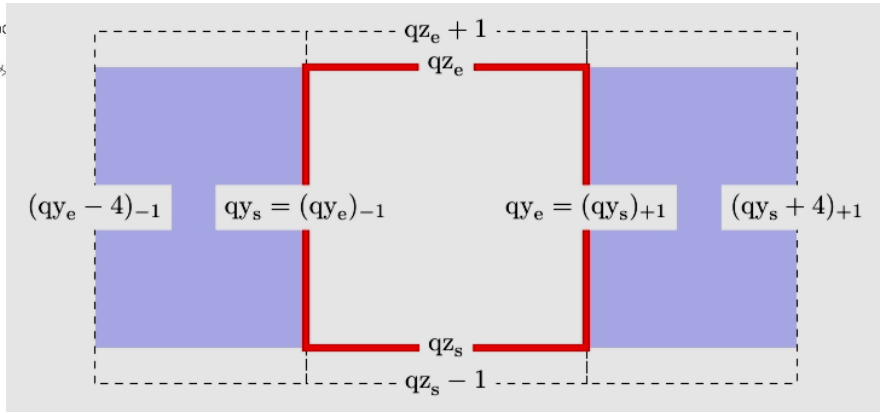
Hybrid MPI/OpenMP parallelism in multidimensional version of VERTEX

VERTEX works on a spherical grid



- **Hydro module**
  fully MPI parallized
- **Transport module**
  OpenMP parallized along ``angular rays´´

Only next neighbour communication
⇒at least 4 hydro zones per MPI-task (ghost zones)



The VERTEX code: programing

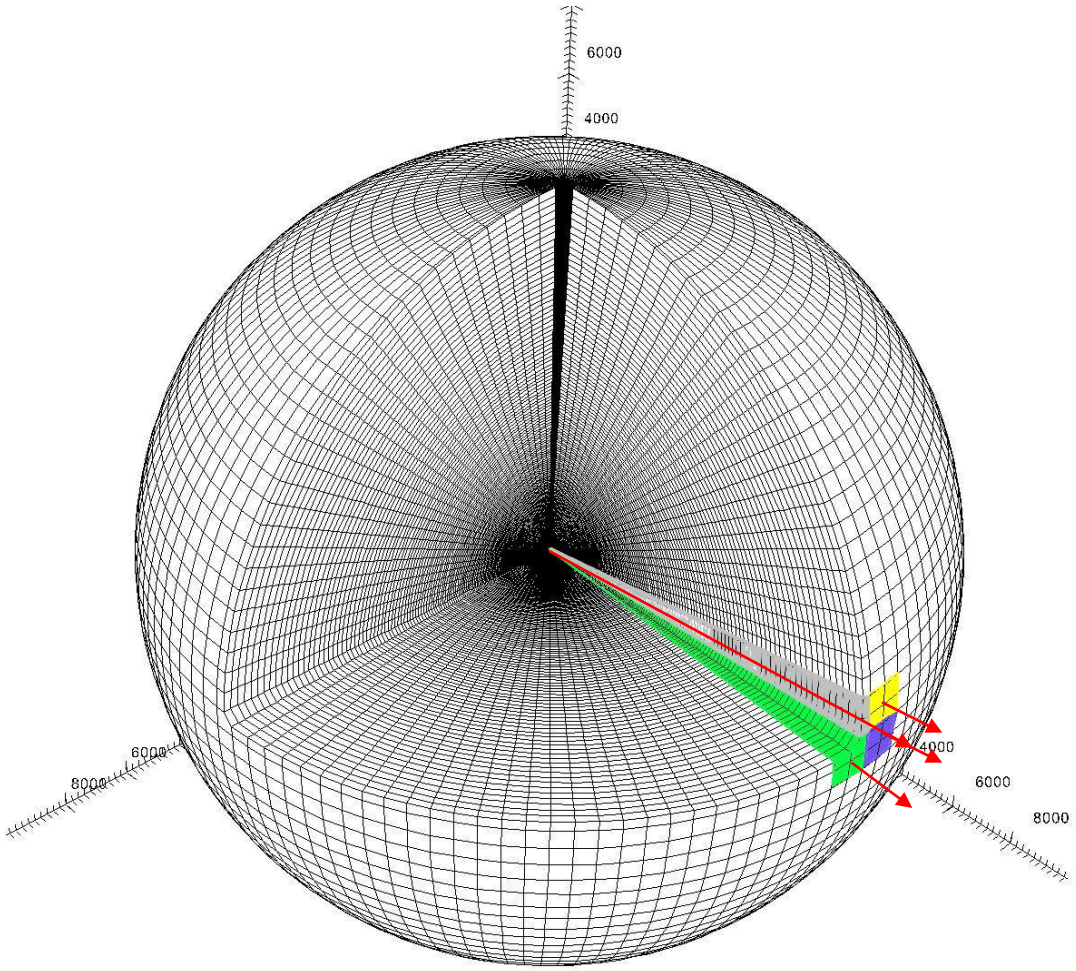
- **Coded in Fortran 95 (and elements of Fortran 2003) and C**

  Our major choice of programming language is Fortran, however, if the need arises we use C

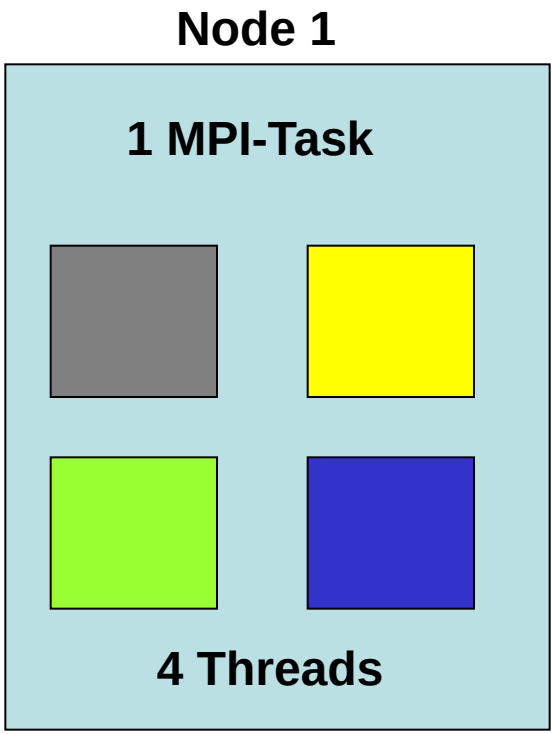
- **we use Fortran/C together with Python unit test**

  at each SVN commit tests are done and depending on them commit is accepted or rejected

Mapping to the processor grid



Hybrid MPI/OpenMP requires SMP mode on BG



per Node: at least 4x4 hydro zones, 1 transport ray per core

The need for high-performance computing

The 1D and 2D version of VERTEX so far ran succesfully on

- **Desktop PC (< 4 cores)**
- **NEC SX5, SX6, SX8, SX9 (< 256 cores)**
- **IBM Power4, Power5, Power6 (< 6200 cores)**
- **SGI Altix 3700, Altix 4700 (< 2048 cores**
- **Linux clusters (< 8000 cores)**

all systems up to now < 10000 cores!

However, a 3D modell with 1° resolution has 64800 transport rays, and needs at least 64000 cores

$\Rightarrow$ Bluegene architecture is at the moment the only option

but VERTEX is very memory hungry…

Porting VERTEX to BLUEGENE/P: memory

Our biggest problem: memory, memory, memory

we started with the need of 1.5 GB/core

First issue: where do we need how much memory?

we instrumented every memory allocation in a module, such that it gives the memory allocated

replace Fortran ``allocate(foo(1:n))´´ -> ``my_allocate(foo(1:n)) ´´ which prints how much memory is allocated

⇒we found arrays that could be resized or removed
⇒ refactoring from time to time seems useful ☺

Determining the memory consumption

Our biggest problem: memory, memory, memory

we still had to know, where how much memory is used,
e.g. in subroutines are there temporary copies created by compiler?

How can you measure the memory needs of a code?

On Bluegene IBM helps here by specific system calls!
(see e.g. www.redbooks.ibm.com/redbooks/pdf/sg247287.pdf)

```
Kernel_GetPersonality( &mybgp, sizeof(_BGP_Personality_t) );
procMB = BGP_Personality_DDRSizeMB( &mybgp);
Kernel_GetMemorySize( KERNEL_MEMSIZE_STACK     , &memory_size);
Kernel_GetMemorySize( KERNEL_MEMSIZE_STACKAVAIL, &memory_size);
Kernel_GetMemorySize( KERNEL_MEMSIZE_HEAP      , &memory_size);
Kernel_GetMemorySize( KERNEL_MEMSIZE_HEAPAVAIL , &memory_size);
```

⇒ **we wrote a library to instrument any part of the code and get the memory usage in this segment and performed a detailed analysis**

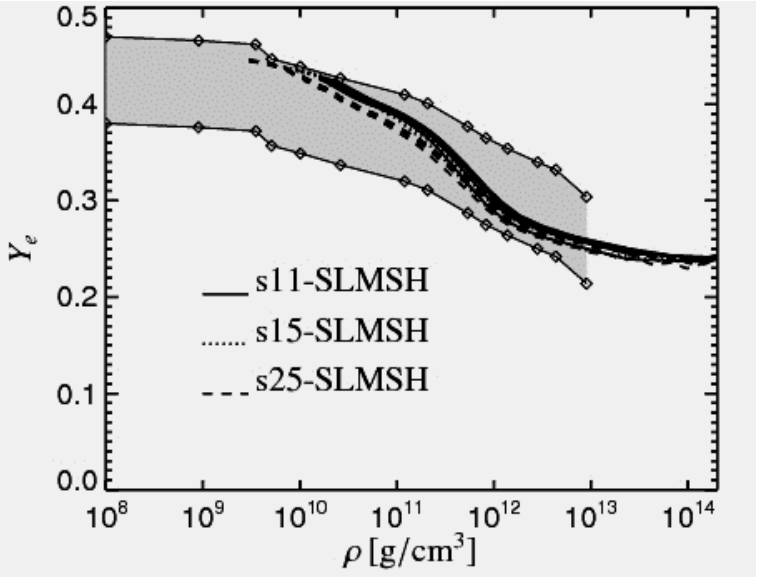
Handling of look-up tables

Vertex needs look-up tables (e.g. for determining state of NS matter)

for state variables (density, temperature, composition) values are looked-up and interpolated from table

Problem: size of tables ~ 300 Mb (per node)

but: not all table data points are needed at all times, i.e the ``trajectories´´ evolve through the tables



At one timestep ~ 1% to 4% of table data is needed

Solution: dynamically load table as envelope around trajectories

Handling of look-up tables (cont.)

Solution: dynamically load table as envelope around trajectories

On sytems with virtual memory, that is easy!

Replace ``normal´´ Fortran read instruction with C mmap call:

```
mem = mmap(NULL, stat.st_size, PROT_READ, MAP_SHARED, fd, 0);
```

However, system call ``mmap´´ limited, since Bluegene does not support virtual memory

=> We had to program a dynamical table loader by hand ☹

This required Fortran C-bindings, since tables are best handled in C

Handling of look-up tables (cont.)

Sketch of dynamic loader:

```
! check whether the given points are still located in currently loaded sub-table
out_subtable(:) = out_of_local_subtable()
if (any(out_subtable(:)) then
   ! set semaphore lock: we do not want to reload table if a thread is in it
   call omp_set_nest_lock()
   ! determine new necessary sub-table dimensions
   call determine_subtable_dimensions()
   ! unload sub-table
   err = flush_subtable()
   ! reload sub-table
   call load_subtable()
   ! remove semaphore lock
   call omp_unset_nest_lock()
endif

 continue with evaluating the table and obtain look-up values
```

Saving memory by using pointers

OpenMP parallelisation of rays is natural to implement with local copies

```
module data
  real, allocatable :: rimlag(:,:,:,:,:,:)
  real, allocatable :: rim     (:,:,:,:)
!$omp threadprivate(rim)
end module
```

O(100 Mb)

O(25 Mb) per thread

both arrays exist twice!
~400 Mb = 20 % of memory per node

```
subroutine get_sect(j,k)
! get local data of ray j,k to work on
do i=1,isma
   do ie=1,iemax
     do i=0,iemax
       do kk=cmin,i
         rim(i,kk,ie,is) = rimlag(i,kk,ie,is,j,k)
       enddo
     enddo
   enddo
  enddo
```

stores global intensity

stores intensity on ray

get ray data and later copy back

Saving memory by using pointers

OpenMP parallelisation of rays is natural to implement with local copies

```
module data
  real, allocatable, target :: rimlag(:,:,:,:,:,:)
  real, pointer :: rim     (:,:,:,:)
!$omp threadprivate(rim)
end module
```

O(100 Mb)

O(0 Mb)

200 Mb per node
=> 50 % savings!

```
subroutine get_sect(j,k)
! get local data of ray j,k to work on
rim => rimlag(:,:,:,:,j,k)
```

This only works if compiler does not introduce a temporary copy!

Luckily on BLUEGENE it works ☺

Saving memory: using parallel HDF5

Formerly, all data was gathered by proc = 0 and then written to file (communication and memory intensive) using parallel HDF5 solved that problem…

But writing data an petaflop machines will be a problem

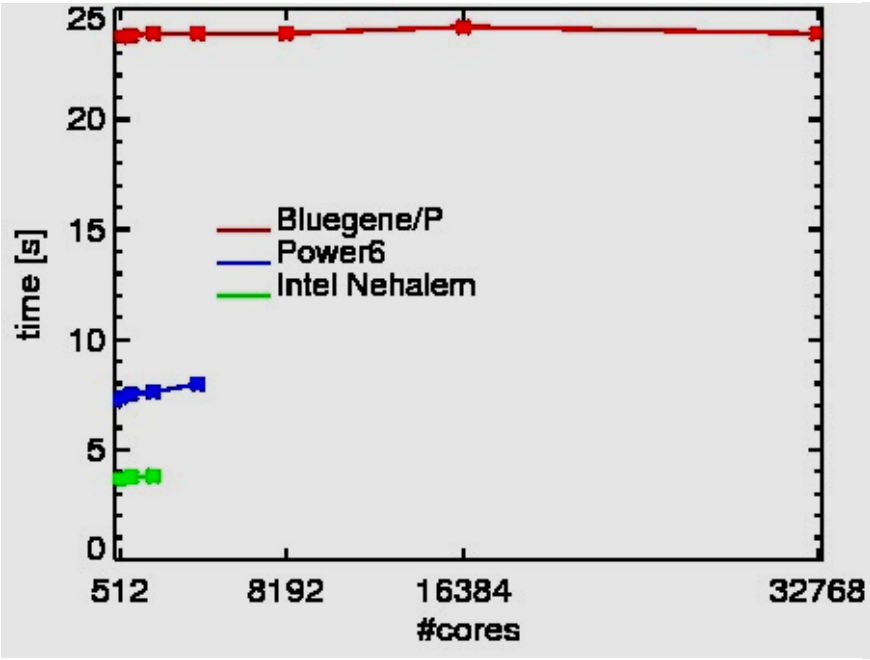
#cores	512	1024	2048	4096	8192	16384
Bandwidth MB/s	822	746	702	728	785	776

A typical setup on 16k cores would write 80 GB output files and 500 GB restart files ~103 to 512 s for write

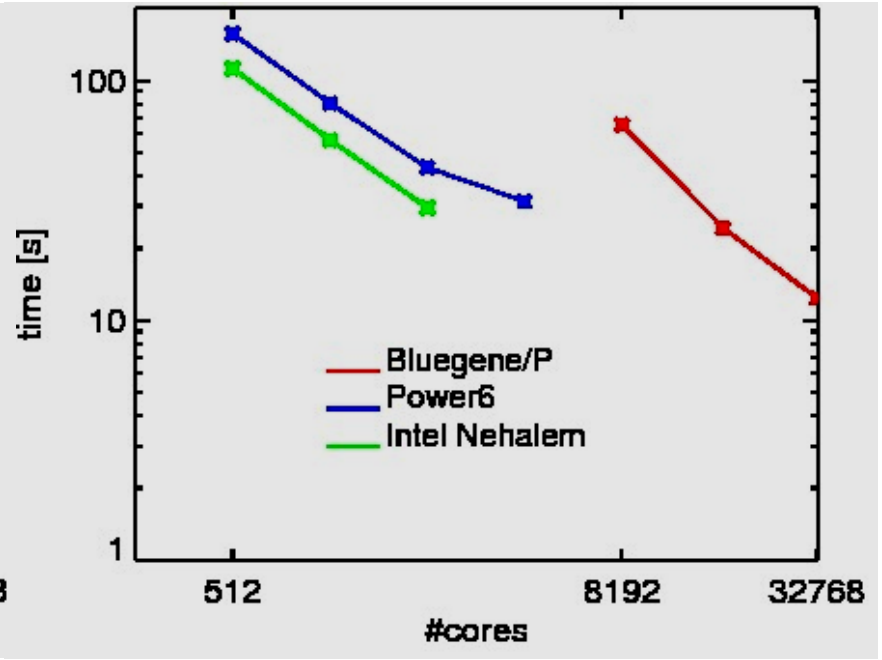
A 1° resolution run (> 65k cores) would already need ~400 to 2000 s for write (and only if writing scales) ☹

Scaling behaviour

Weak scaling



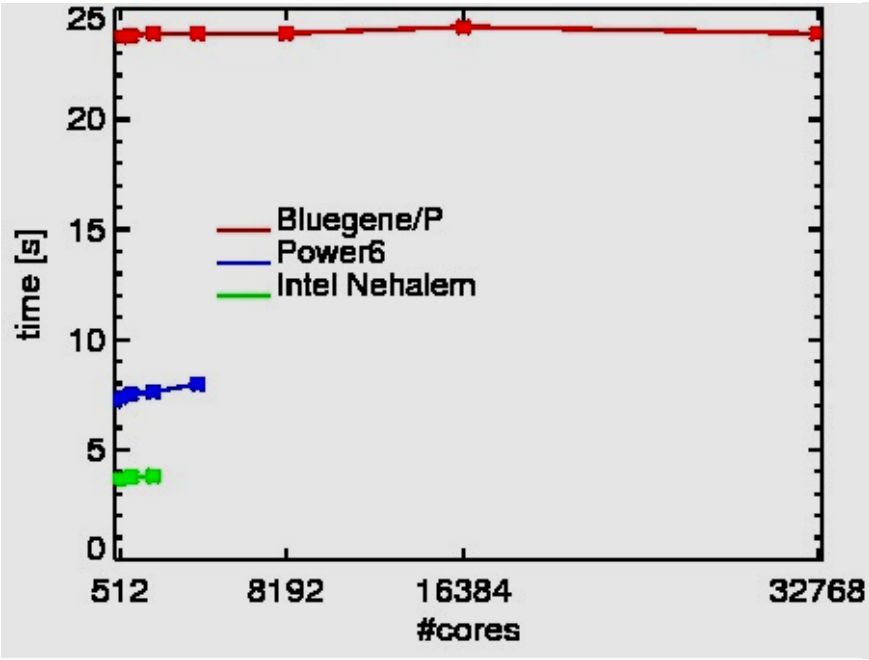
Strong scaling



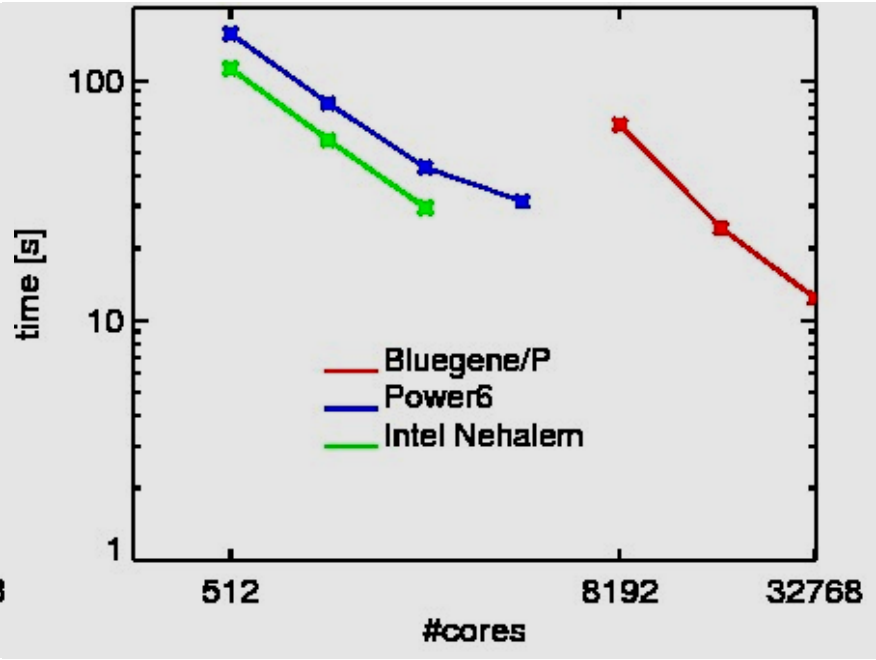
Up to know we were able to run on 32k Bluegene cores, and on Power6 and Linux clusters of much smaller size

Scaling behaviour

Weak scaling



Strong scaling



We had one chance to test on higher core numbers for ~30 min.
Code hang up: redirect every stdout , stderr to a single file ☹

Debuging on Bluegene

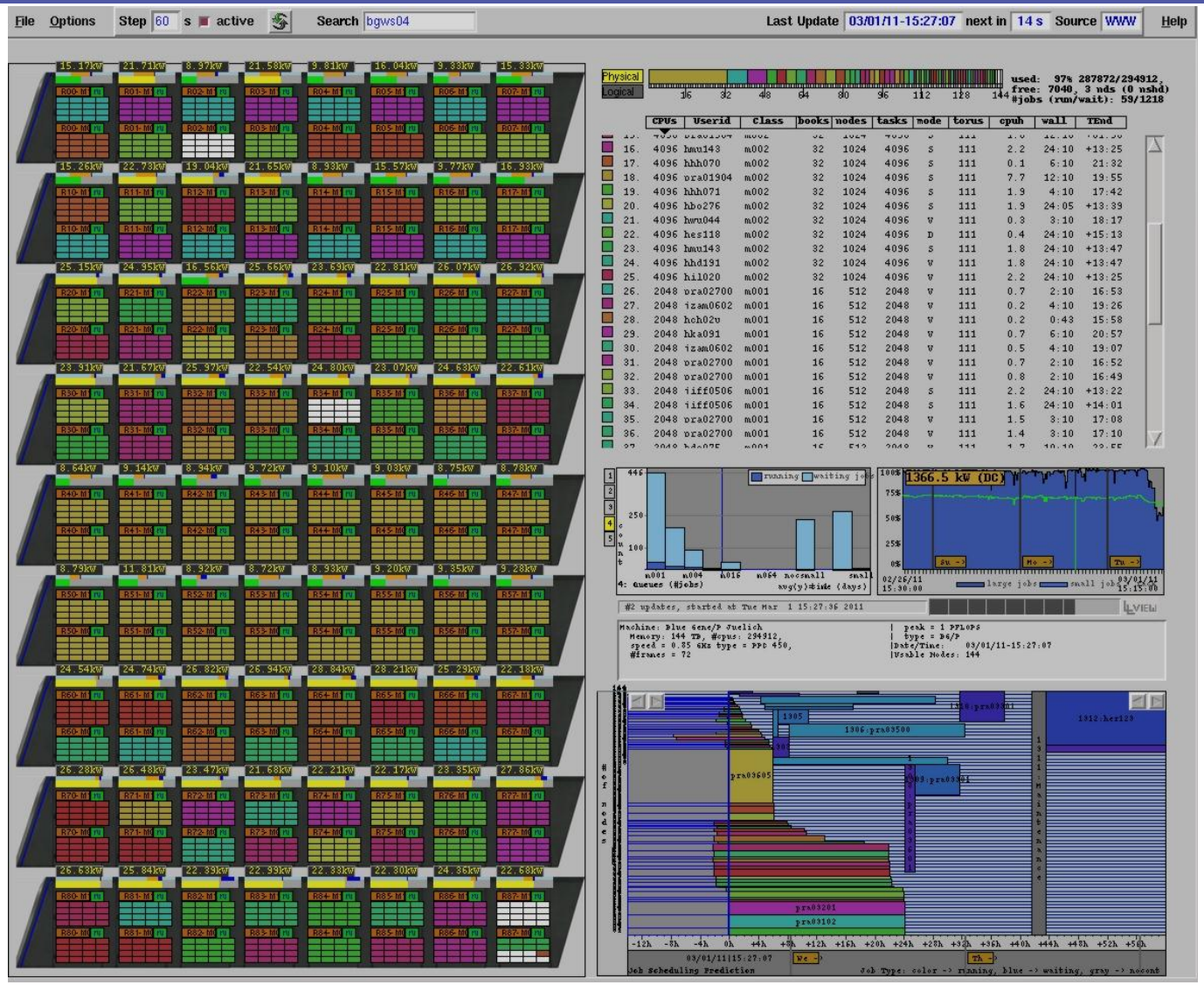
It`s a bit cumbersome:

- large numer of cores
- debuggers available?
- long times for partitioning
- lack of possibility to test on single cores
- turn-around time for > 8k

Special features of Bluegene:

- core files are simple ascii-files
- addr2line tool is useful
- IBM-coreprocessor tool quite usefull:
  /bgsys/drivers/ppcfloor/tools/coreprocessor

Getting a Bluegene overwiew: llview



Developed at Research center Juelich

Analyzing MPI-communications

We found IBM libmpitrace extremly helpfull to get a feeling for the MPI-communication and to analyze it:

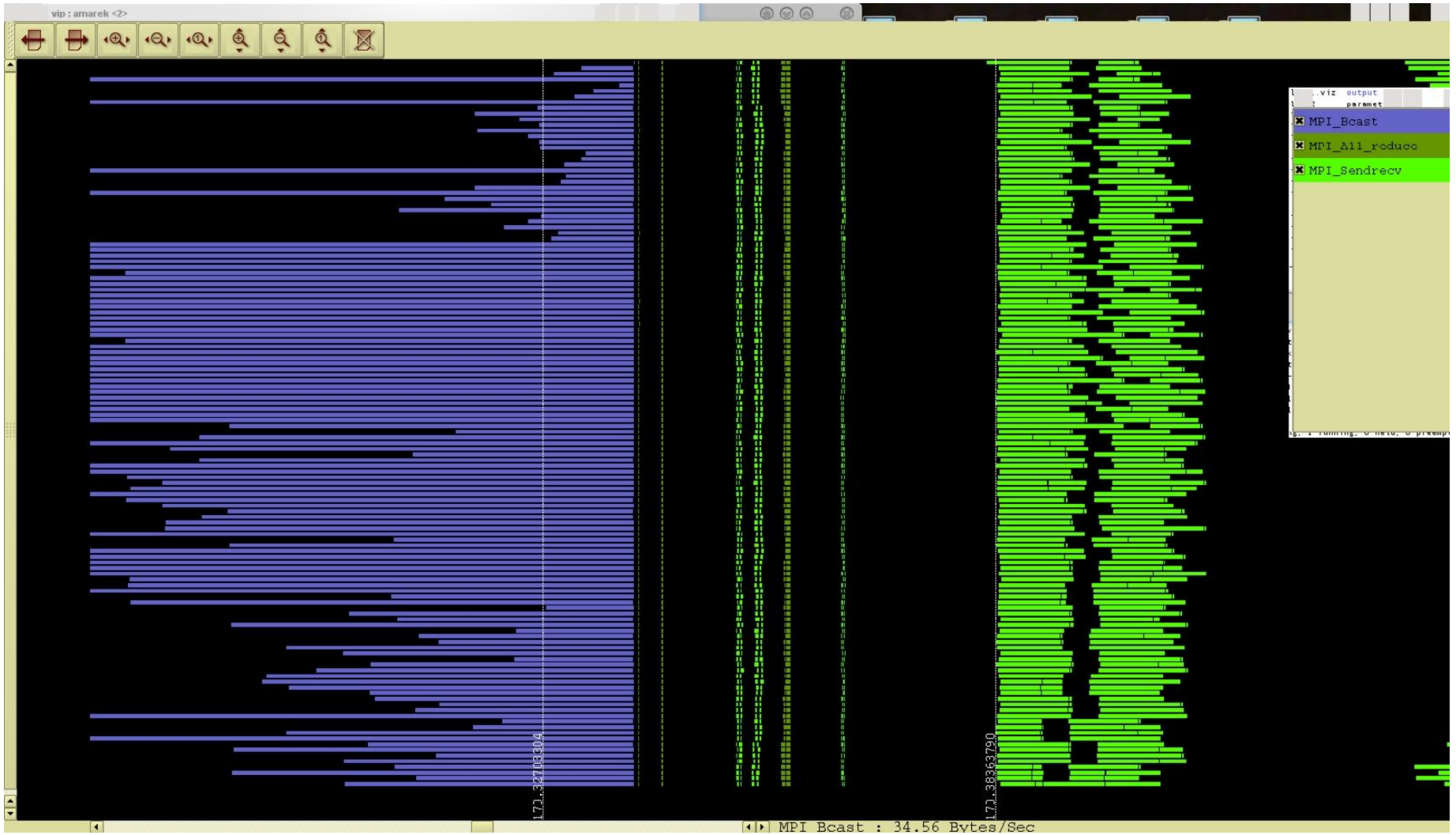
It gives the statistics of used communication

- **message lengths**
- **min/max/median communication times**

and detailed information per task (if you want)

a graphical tool (from IBMs hpctoolkit) helps to interpret the data

Analyzing MPI-communications



Going beyond 32k cores

We were able to participate in the „Juelich Extreme Scaling Workshop 2011" and to do measurements on a Bluegene/P up to 300k cores

On this size it is a whole new game and problems become visible:

- **Partitioning of system (can take up to 40 mins!)**
- **Communication that works fine on <32k cores suddenly hangs**
- **debugging even worse ☹**
- **wall-clock time is an issue**

We were not able to run the full code, but instead to measure different subroutines, kernels etc.

Issues with Bluegene/P

- The memory per core (500 Mb) is very limiting
- The small number of cores per node is a problem with VERTEX
  Remember: we need at least 4x4 zones per MPI task (=node)
  => the fraction of ghost-zones to compute zones is larger on BG than on a 8 cores/node system
- The low frequency of cores, though energy efficient, is a limiting factor: One needs ~ 8 times the #cores of a Intel Nehalem system
- No single-core available to check optimizations

Speculations: the future

Systems with #cores O(100000) will have to be energy efficient

=> Bluegene architecture will be more and more important

Bluegene/Q (from what one can hear at the moment)

- **18 cores per node (favoured by VERTEX)**
- **~ 1.7 GHz frequency => 2x times faster than Bluegene/P (helps every application)**
- **propably at least 1GB memory per core. Wow, that really helps**

Summary

- **The VERTEX-code has been recently modified to run on massively parallel computer systems**
- **IBMs Bluegene system is at the moment the only system available to us with a sufficient number of cores**
  - **the small amount of memory per core turned out to be a though problem**
    - ⇒ we measured (detailed) the memory needs of VERTEX
    - ⇒ we reduced the memory consumption by changing the memory layout
    - ⇒ we also introduced a dynamical table loader
    - ⇒ The usage of pointers als helped
  - **we analyzed the MPI communication pattern and found some unfavorable code**
  - **at the moment we achieve good scaling on up to 32k cores**
- **we are working on a still better scalable version and hope for computing time on up-coming HPC systems**

Summary cont´d

- beyond 32k cores everything becomes cumbersome…
- at each new number of cores unforseen problems may arise, which require new debuging

=> ``wish-list´´ for application enabeling on this size

- easier debuging ☺ But how?
- easy access to needed core size in a fast turn-around way
- in case of Bluegene: lower partitioning times

Questions ?