A Parallel Code for Kinetic 3D Lattice MC Simulation of Nucleation, Growth and Ostwald Ripening of Nanocrystals

Nils Schmeißer
Dr. Manfred Kunicke
Dr. Karl-Heinz Heinig

Contents

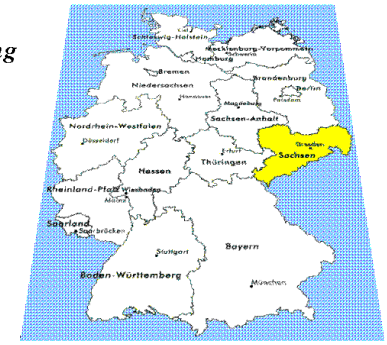
- about Forschungszentrum Rossendorf
- the physical process
- the need for speed-up
- considerations on parallelization
  - three approaches
  - the development process
- results
- conclusions/problems
- outlook

Forschungszentrum Rossendorf e.V.

Institute of Ion-Beam Physics and Materials Research
Institute of Bio-anorganic and Radiopharmaceutical Chemistry
Institute of Nuclear and Hadron Physics
Institute of Radiochemistry
Institute of Safety Research

Technical Infrastructure:
Department of Communication and Data Processing

URL: http://www.fz-rossendorf.de



Dep. of Com. & Data Processing

Hardware:

- CONVEX C3220 - login server, print server (1998 out of services)
- CONVEX C3820 - data storage (UNITREE) and vector computing
- CONVEX SPP1000/CD-4N, SPP-UX 3.2.129 - parallel & serial computing
- CONVEX/HP SPP2000/S-Class, SPP-UX 5.2.1.14 - parallel & serial computing
- 3 * IBM RISC 6000 - computing and several services
- 4 * HP 9000 - several sevices

The Physical Process 1

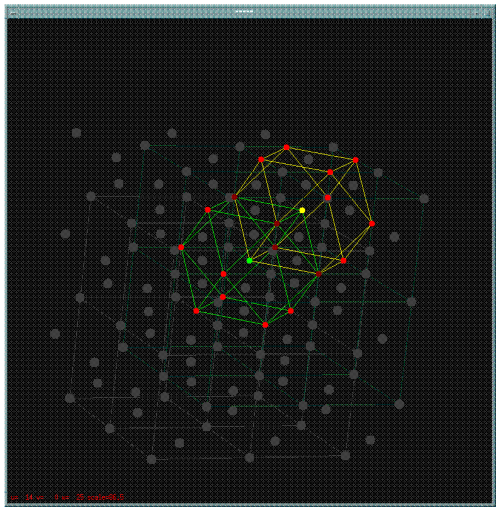
- ion implantation can be used to manipulate properties of solids or thin films
- the Ion Beam Synthesis (IBS) process consists of:
  - supersaturation of the matrix by implanted impurity atoms
  - nucleation of nanoclusters
  - growth of nanoclusters
  - subsequent Ostwald ripening (OR)
  - coalescence (for high doses)

The Physical Process 2

- screen-shots from the video

The Physical Process 3

- atomic scale process
- the transition behaviour of implanted ions is modeled by determing their transition probability out of the constellation of their nearest neighbours



First Simulations 1

- physical relevance:
  - this really complex system can be investigated only by numerical simulation
  - within this model self-organisation of nanoclusters can be demonstrated
- simulation method
  - Monte-Carlo simulation allows to study all subprocesses of IBS
  - atomic interactions are simulated by a cellular automaton approach

First Simulations 2

- the MC algorithm was originally coded in Pascal and run on IBM-PC *
- the code was highly optimized for this architecture
- a production run (lattice size 256^3, step by step implantation of $4*10^4$ atoms and additional ripening steps) took about 24 hours

* K.-H. Heinig

Why Speed-Up is Needed

- for experimental setup, the operator requires time to solution <10 hours
- large scale prediction is needed
  - lattice size $\geq 1024^3$, for silicon this corresponds to the size of a 0.18 µm structure device
  - number of implanted atoms in the order of 10^6
  - number of simulation steps >> number of implanted atoms

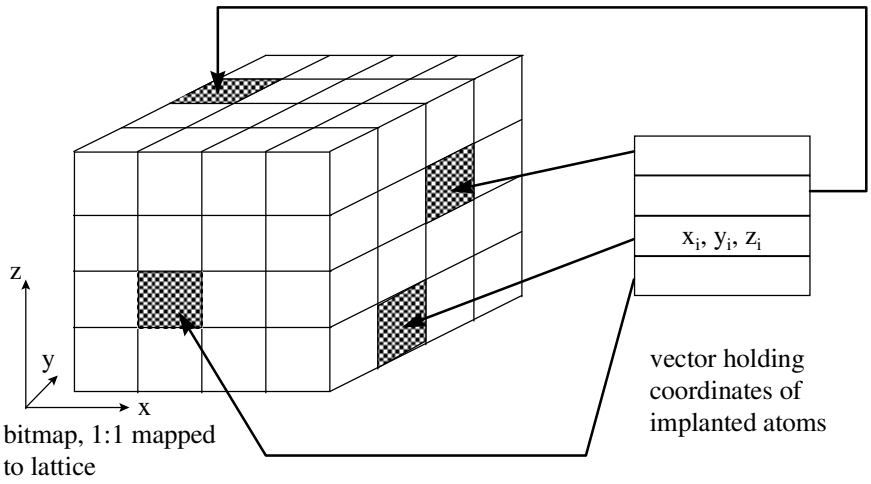
Porting to Workstation

- porting Pascal code to C gives the ability to let it run on workstations
- transformation was done 1:1
- running on HP PA-RISC 8000 was not much faster than on an Intel Pentium
- to get a really large speed-up the code has to be parallelized

Double Bookkeeping Technique 1

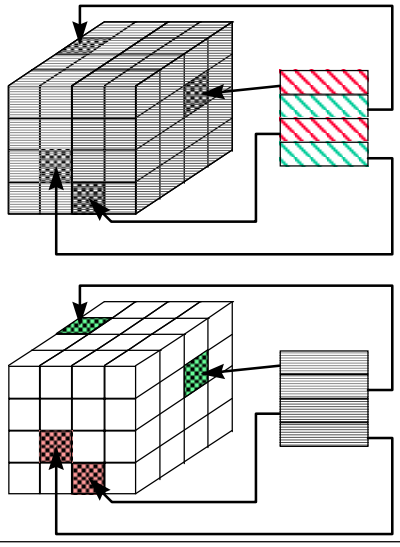
- the coordinates of the implanted atoms are
  - stored in a vector and
  - marked in a bitmap representing the lattice-space (1 bit corresponds to one place in the lattice)
- fast access to the configuration around a certain place in the lattice
- the double bookkeeping technique was almost used in the original code

Double Bookkeeping Technique 2



How to Parallelize

- two main ideas
  - distributing the lattice space
  - distributing the particle space



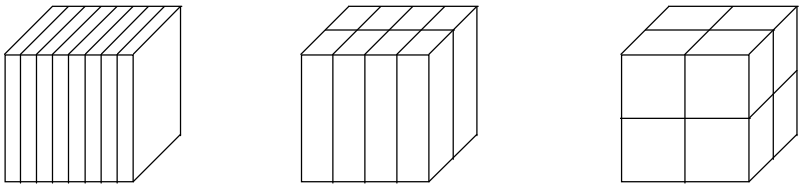
Distributing Lattice Space 1

- a processor is assigned to a subset of the lattice space
- the algorithm
  - each atom inside a certain subspace is handled by the assigned processor
  - if an atom leaves a subspace the processor has to notify the processor assigned to the neighboured subspace
  - **problem**: if an atom is inside a 1-place depth border of a subspace, the neighboured processors has to be informed too

Distributing Lattice Space 2

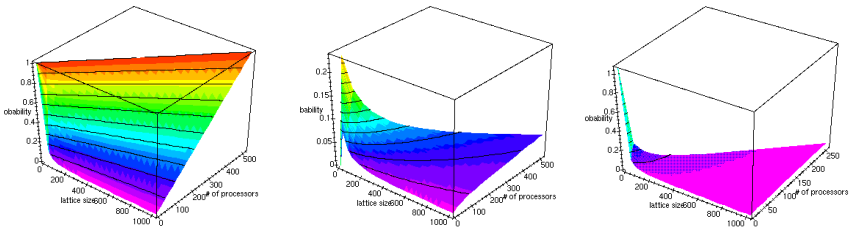
- for efficiency the probability of being inside a border of depth 1 needs to be low
- the estimated probabilities for an atom to be inside a border of depth 1 of a s^3 lattice are:

$$P = 2\frac{2s\cdot(s-1)+n_p\cdot(s-2)^2}{s^3} \qquad P = 2\frac{(s-2)\cdot\left(s+(s-2)\sqrt{n_p}\right)}{s^3} \qquad P = 2\cdot\sqrt[3]{n_p}\cdot\frac{2\cdot\left(s^2-\sqrt[3]{n_p}\right)+\left(s-2\cdot\sqrt[3]{n_p}\right)^2}{s^3}$$



Distributing Lattice Space 3

- if the lattice is large enough and the number of processors is small enough the event an atom is inside the 1-place depth border occurs really seldom ($p \sim s^{-1}$)



Distributing Lattice Space 4

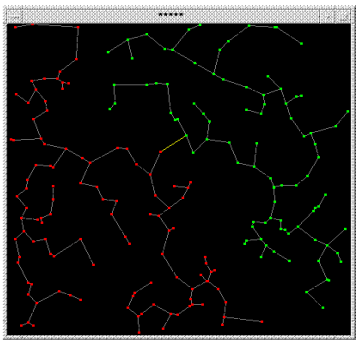
- but: during the process atoms begin to organize in clusters
- this leads to inhomogeneous atom densities in the lattice-space and causes therefore load-imbalances

Distributing Particle Space I-1

- graph-theoretical approach
  - dividing the set of atoms into two subsets where:
    - the distance between the subsets is maximized
    - the cardinal of both subsets are nearly the same
  - taking the complete distance graph and finding the minimum skeleton within gives the solution
    - an algorithm by KRUSKAL can be used to find the minimum skeleton
    - a weight function is introduced to find the edge which divides the two subsets (allows control of importance of the cardinal balance and the area which is searched)

Distributing Particle Space I-2

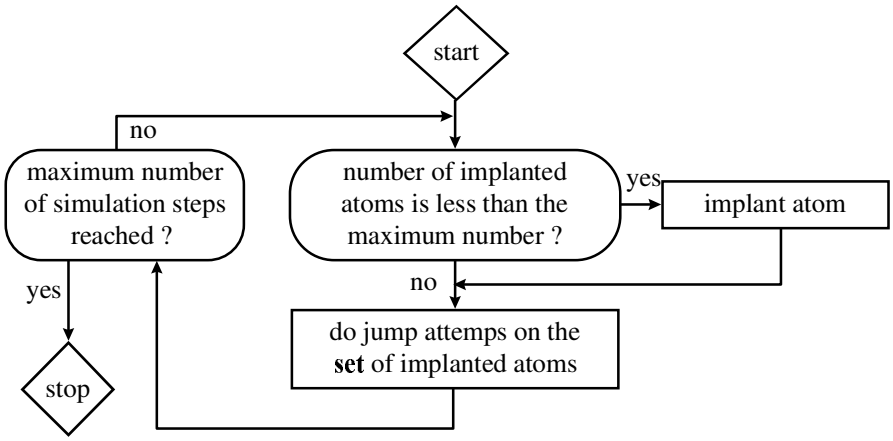
- Result:
  - works fine
  - requires $O(n_{atom}^3)$
- can be used if the average distance between two atom subsets is large ($\geq O(n_{atom}^2)$)



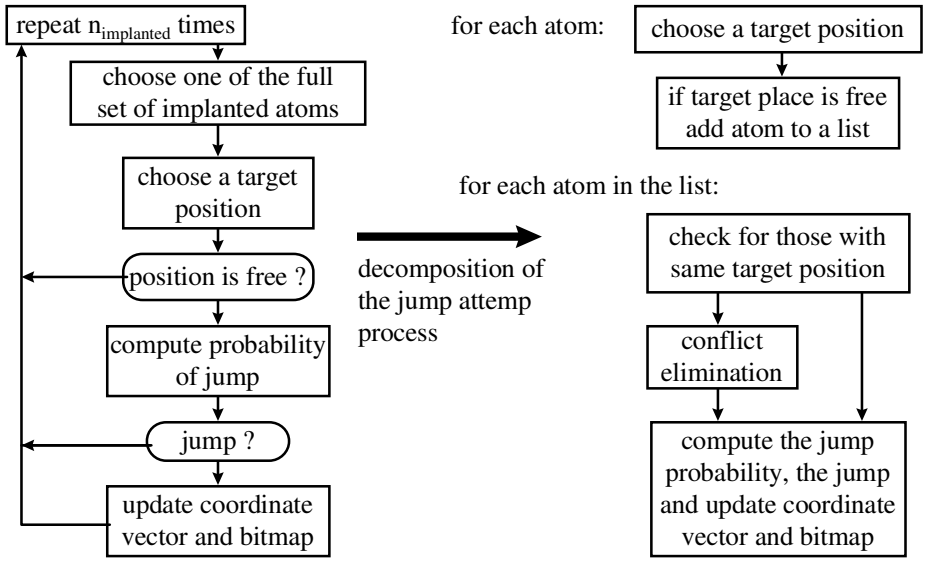
- during implantation and the first ripening steps the average distance equals $s_{lattice}*(3/n_{atom})^{1/3}$

Distributing Particle Space II-1

- approach: re-formulating the physical model



Distributing Particle Space II-2



Distributing Particle Space II-3

- the “old” model allows jump cascade queing with very low probability ($n^{-cascade_steps}$)
- excluding cascaded jumps doesn’t change essentially the underlying physics
- conclusions:
  - the problem itself is inherent parallel
  - the problem consists of three main steps
    - propagating jump destination for each particle
    - eliminate conflicts
    - compute jump attemps

Distributing Particle Space II-4

- propagating jump destination
  - each implanted atom can potentially jump to a new free place in the lattice
  - the one out of 12 directions an atom could leave its origin is equally distributed
  - if the destination, pointed to by the randomly choosen direction is not free, the atom will not jump and need not to be considered further
  - the propagation for any atom is not depending on the behavior of the others

Distributing Particel Space II-5

- conflict elimination
  - in some cases the predicted jump destinations can be the same one for different atoms
  - this computational model artifact needs to be resolved
  - this can be done by randomly choosing one of those atoms
  - the selection needs to be equally distributed

Distributing Particle Space II-6

- jump attemps
  - collecting information on the effective nearest neighbours for the subset of choosen implanted atoms
  - from the neighbours constellation the binding energy and finally the transition probability is evaluated

Implementing the new idea

- the following tasks have to be solved
  - choose a paradigm of parallelization
    message passing / shared memory
  - implement steps 1 and 3 in parallel
  - do optimization
    depends on run-time analysis of the code written so far

Jump Propagation

- as described in the model, this part is reduced to a simple parallel loop (pseudo code)

```
integer imp(3,MAXIMP)     ! actual position
integer impn(3,MAXIMP)    ! propagated position
integer dir(MAXIMP)       ! jump direction
c$dir loop_parallel
do i=1,nimp
  jflag[i]=.false.
  dir(i)=Random(1,12)        ! Mlib call
  impn(1:3,i)=imp(1:3,i)+rotate(1:3,dir(i))
  if (.not. BitmapGet(impn(1:3,i)) then
    jflag[i]=.true.
  end if
end do
```

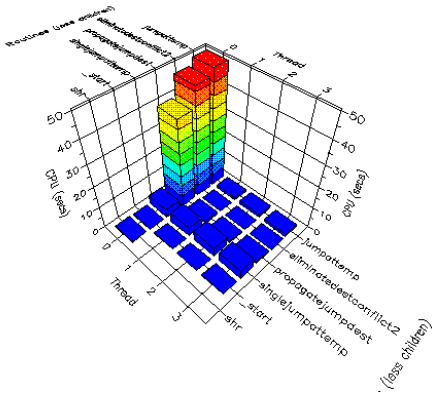
Jump Attemp

- the atoms which get a jump attemp are indexed by the field **`index`**

```
c$dir loop_parallel
do k=1,nindex
  i=index(k)
  p=imp(1:3,i)
  if (SingleJumpAttemp(p,dir(i))) then
    pn=impn(1:3,i)
    c$dir critical_section
      UpdateBitmap(p,pn)
      imp(1:3,i)=pn
    c$dir end_critical_section
  endif
enddo
```

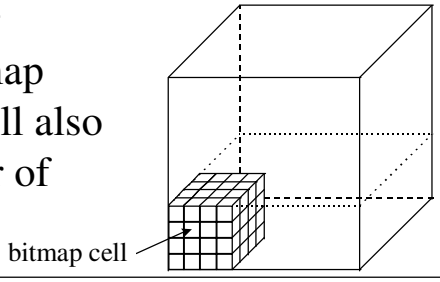

Run-time Analysis

- bad load-balance
- poor performance
- bad scaling
- reason:
  - huge amount of cache misses



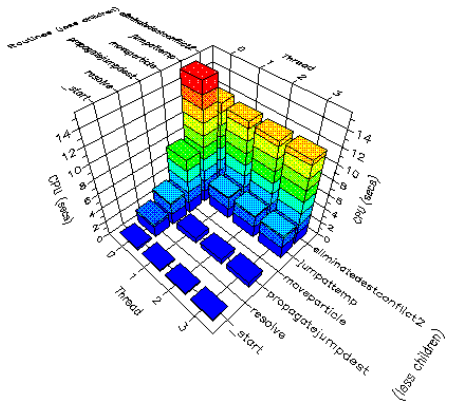
Cache Optimization

- organizing the coordinate vector as integer(8,i), holding both, old and propagated position, as well as the direction of the jump
  - integer(8) fits to a cache line; the number of cache misses in the subroutine jump_attemp can be reduced by factor 4
- organizing the bitmap as 4*4*4 bitmap cell also reduces the number of cache misses



Run-time Analysis

- the time, spent in the jump_attemp subroutine decreased
- there is still an important amount of sequential work done (again due to cache misses)

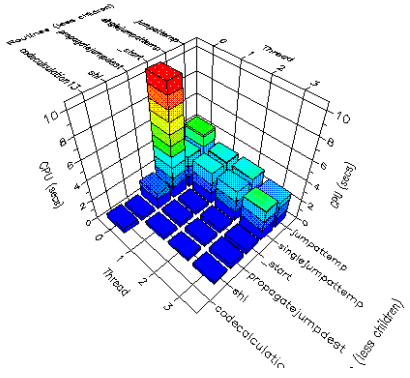


Optimizing Jump Attemp

- the critical section caused a lot of cache misses (large latency!)
- how to avoid the critical section
  - updating the bitmap is critical
  - each processor updates those cells which index hold $index \mod n_{processors} = i_{this_processor}$
  - in most cases this distribution will not produce a load imbalance

Run-time Analysis

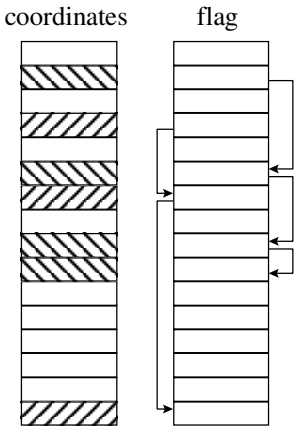
- after eliminating the need for a critical section, the jump_attemp subroutine scales as expected
- a significant amount of time is spent in conflict elimination



Conflict Resolving 1

- became important **after** parallelization and optimization of steps 1 and 3
- steps to be done for resolving conflicts
  - finding all subsets of atoms, jumping to the same place and consisting of more than one atom (can be done parallel)
  - for each of those subsets choosing one of their members and remove the others from the list of further considered atoms (can be done independently for each subset)

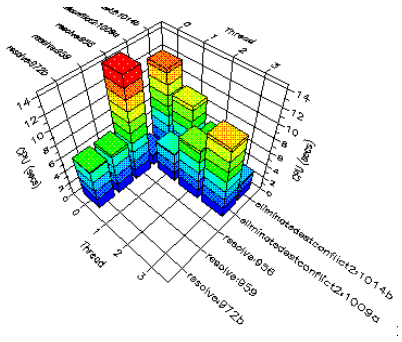
Conflict Resolving 2



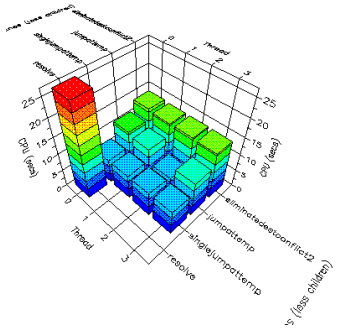

```
c$dir loop_parallel
do i=1,nimp
  p=imp(4:6,i)
  do j=i+1,nimp
    if (flag(i). eq. 0) then
      if (p .eq. imp(4:6,j)) then
        flag(i)=j
        confl=.true.
      endif
    endif
  enddo
enddo
```

Conflict Resolving 3

before

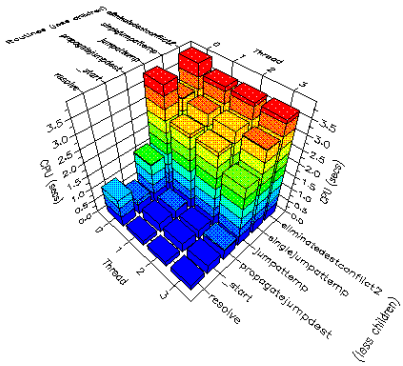


after changing chunk size to 4



Run-time Analysis

- this CPU-time measurement present the final state of the code, after the resolve subroutine was parallelized too
- except some small fluctuations it shows a good load-balance



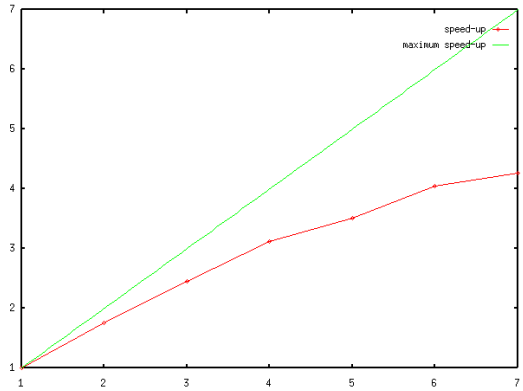
Results

- we will demonstrate speed-up and we'll discuss the quality of these results

- test job:
  - minor problem is (256^3,1000,2000)
  - major problem is (256^3,2000,4000)
  - real problem is (256^3,$4*10^4$,>$4*10^4$)

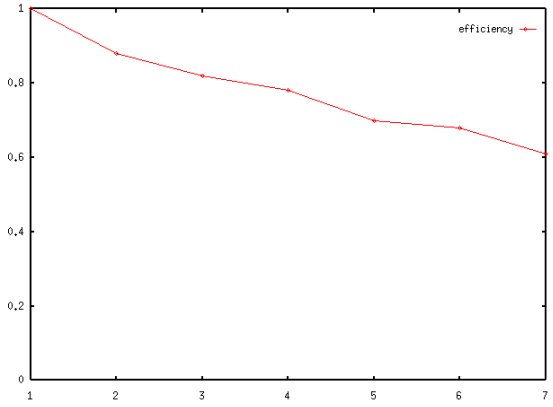
Results - Speed-up

- the application scales not bad
- the sequential portion is still worth to be considered



Results - Efficiency

- a better measure for the quality of the parallel code:



Conclusion

- half a year was required to optimize 600 lines of that code
- the main job was done on re-modeling and cache optimization

Problems

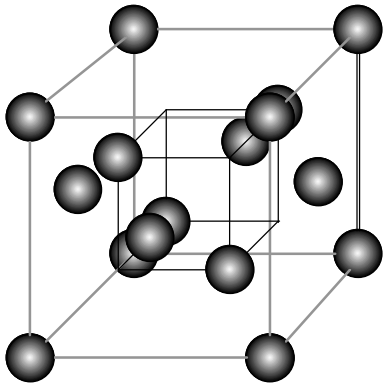
- when doing the performance analysis it turned out, that:
  - the performance results are unstable, this might be due to system inconsistences
  - the cxpa tool (4.2.129) does not work properly
    - time measurements are influenced by the selected profile
    - the 3D profile of cache misses for parallel loops doesn't work

Outlook

- production of test data
- check-point & restart
- if wanted porting to message-passing paradigm

in the future we wish to get:

- an exact cache performance analysis
- a native Fortran90 compiler
- a native High Performance Fortran Compiler
- sophisticated C/C++/HPC++ compilers

The Physical Process 3

- atomic scale process
- the transition behaviour of implanted ions is modeled by determing their transition probability out of the constellation of their nearest neighbours