COMPUTER SIMULATIONS OF HIGH-ENERGY HEAVY ION COLLISIONS

By

Gerd Kortemeyer

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Physics and Astronomy

1997

ABSTRACT

COMPUTER SIMULATIONS OF HIGH-ENERGY HEAVY ION COLLISIONS

By

Gerd Kortemeyer

One of the still most challenging questions in nuclear physics is that of the equation of state (EOS) of nuclear matter – how does nuclear matter change its properties under different temperatures and pressures? Is there a phase transition to quark gluon plasma? Heavy-ion collisions are of great importance in this study; when two large nuclei collide, the nuclear matter is compressed, and due to two-body collisions of the nuclear constituents the temperature rises. Unlike for macroscopic matter, for nuclear matter this state is unfortunately not directly observable: both the length and the time scale of nuclear collisions prohibit probing nuclear matter while it is still at high temperature and density. Instead, the nuclear matter already cooled, expanded, and formed more stable configurations before detection.

However, these late reaction products still carry information that helps to reconstruct what happened in the early stages of the collision. It is the task of simulations to aid the reconstruction of the reaction mechanism and conditions from the late reaction products.

In this thesis, computer simulation tools for nuclear reactions are used to examine the consequences of different microscopic models.

To my wife Anna

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Intermediate Energy

1.1.1 Percolation Models

In intermediate energy heavy-ion collisions, the production of complex fragments still is a puzzling aspect of nuclear physics research. Experiments have been conducted both with proton and with heavy-ion beams, and several seemingly different phenomena, such as evaporation, spallation, fission and multi-fragmentation have been observed.

In multi-fragmentation scenarios, the hot nuclear system tends to decay into multiple intermediate mass ($3 \leq Z \leq 20$) fragments (IMFs). Significant differences in the relationships between fragment, neutron, and charged particle multiplicities were found between ^{112}Sn+^{112}Sn and ^{124}Sn+^{124}Sn collisions at 40 MeV/A [Kun96]. The results are incompatible with a universal scaling of the average number of intermediate mass fragments versus the number of charged particles and neutrons, respectively, and in Chapter 2, the possibility to explain this phenomenon in the framework of percolation models is explored. It was found that the results are only reproducible in part, which indicates contributions of nuclear phenomena beyond geometrical fragmentation schemes.

1.1.2 Transport Models

For a long time, simulations of heavy-ion collisions have been rather phenomenological in nature, which somewhat corresponded to early experiments being conducted in an inclusive way – phenomenological models, even if based on different assumptions, would predict the inclusive data with oftentimes almost equal quality. As experiments became more sophisticated and exclusive, the need for advanced simulation methods arose. This was when microscopic models became more popular: one has to be able to dynamically simulate the collisions without any assumption concerning thermal equilibrium – transport models are one of those microscopic dynamical models.

Boltzmann-Uehling-Uhlenbeck (BUU) (also called Landau-Vlasov) codes are semiclassical simulations, in which as an ansatz for the Wigner-function a product of Delta-functions is used. Each Delta-function can be associated with a localized "testparticle." Inserting this ansatz into the Boltzmann equation results in equations of motion for the testparticles that are governed by two terms: a smooth modification of the trajectories resulting from a mean-field term, and a collision term that describes two-particle collisions between testparticles. The actual solution of the Boltzmann equation is achieved by evolving the testparticle essemble in phase-space according to the equation of motion rather than solving a partial differential equation. Chapter 3 gives a more detailed overview of this mechanism.

The semiclassical transport codes work reliably at energies beyond 100 MeV per nucleon. Below this energy, quantum effects are too pronounced to be neglected. The BUU codes followed earlier transport codes, so-called cascades,[Cug81] which did not have a mean-field term yet. This approximation is only appropriate at even higher energies, above around 1 GeV per nucleon (Appendix A describes the details of a cascade code). At lower energies, many two-body collisions are blocked due to phase-space considerations, so that the mean-field term becomes dominant. At higher energies, Pauli-blocking is not so much of an issue anymore, since a large phase-space volume is available, which strengthens the influence of the collision term. At the same time, because the particles have larger momenta, the mean-field becomes less influential.

Transport models have successfully described many aspects of intermediate energy heavy-ion collision dynamics. One of the observables in heavy-ion collisions is the collective flow of the reaction products, it contains information about the compressibility of nuclear matter. However, it was indicated [Ale95] that within current implementations of BUU simulations the equation of state is not implemented in a consistent way. In Chapter 3, a hard core equation of state for the nuclear matter is implemented and its effect on the simulated collective flow is explored.

1.2 Ultrarelativistic Regime

1.2.1 Experimental Efforts

Currently, two colliders are planned for the acceleration of heavy-ions to ultra-relativistic energies. The Relativistic Heavy-ion Collider (RHIC), presently under construction at Brookhaven National Laboratory in New York, is a dedicated heavy-ion collider planned for experiments in 1999. RHIC will accelerate and collide ions from protons to heavy nuclei, such as Au, at c.m. energies up to 500 GeV for protons and 100 GeV per nucleon for Au+Au collisions. The luminosity for Au+Au will be 2 x 10^{26} cm^{-2} s^{-1}. Near head-on collisions of Au+Au at RHIC are expected to produce from 500 to 1500 charged particles per unit pseudorapidity at midrapidity in a single collision.

Large detector systems are being constructed to analyze the products of these interactions to observe indications of the possible formation of a quark-gluon plasma and a possible chiral phase transition.

Heavy-ion physics research will also be an integral part of the program for the Large Hadron Collider (LHC), to be constructed at CERN, the European Centre for Nuclear Physics, in Geneva, Switzerland. For Pb nuclei, the c.m. energies at the LHC will be 5.4 TeV per nucleon pair with luminosities of 10^{27} cm^{-2}s^{-1}. Predictions for the charged particle densities at the LHC for near head-on collisions of Pb+Pb range from 2000 to 8000 per unit pseudorapidity. The large uncertainty in these numbers arises primarily from the present lack of information on the distributions of soft gluons in nuclei.

1.2.2 Parton-Cascades

It has been indicated that RHIC and LHC run into somewhat of a "theory-vacuum." The interpretation of these complex collisions and the possible generation of a quark-gluon plasma pose a major problem: what are the experimental signatures? Currently, several possible signatures are in discussion:

- Kinematic effects of a first-order phase transition.
- Electromagnetic probes e^+e^-, $\mu^+\mu^-$, γ measure the *early* phase of quark and gluon distributions and content.
- Hadronic probes of deconfinement and chiral symmetry restoration

$$J/\psi \longrightarrow c + \bar{c} \qquad (J/\psi,\ \Psi' \text{ suppression})$$

$$gg \longrightarrow s\bar{s} \qquad (\text{strangeness enhancement})$$

None of these possible signatures have been commonly acknowledged as reliable. In an effort to aid the answering of this question, a theoretical model for the collision processes that goes beyond a phenomenological description must be developed. One possible microscopic approach is to extend the semiclassical transport theory to high-energy physics[Kal93, Sor89.1, Sor89.2, Gei92.1, Gei94.1, Gei92.2, Gei93], which leads to so-called "parton cascade" codes. As the energies increase in these models to the ultrarelativistic regime, Lorentz covariance and causality are not strictly respected. The standard argument is that such effects are not important to final results; but they have not been seriously considered at high energies. In Chapter 4, it is pointed out how and why these happen, how serious of a problem they may be and ways of reducing or eliminating the undesirable effects are suggested. The appendix gives a commented listing of the major sections of the parton cascade code that was developed as part of this thesis.

Chapter 2

Isospin dependent multi-fragmentation in ^{112}Sn + ^{112}Sn and ^{124}Sn + ^{124}Sn collisions

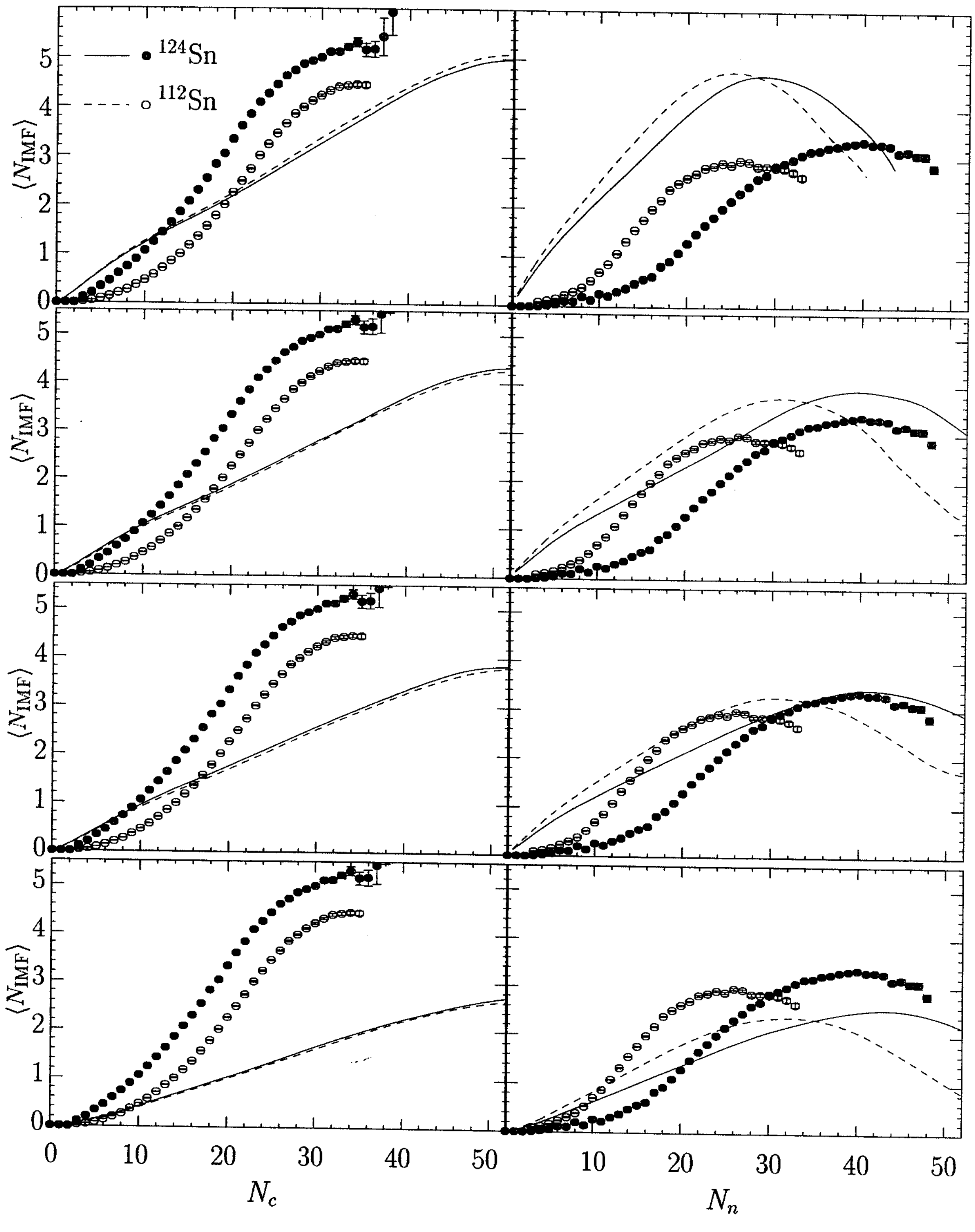
2.1 Introduction

Percolation models have proven highly successful in the simulation of multi-fragmentation reactions in the past [Bau84, Cam85]. Within these models, fragmentation is described by first distributing a set of points or sites, each representing a nucleon, on a 3-dimensional lattice, which represents the bonds between the sites. In the case of a simple rectangular lattice, each site is connected to six nearest neighbors, however, it has been shown that the model is to a large degree independent of the lattice structure [Bau84, Sta79]. In the second step, randomly some lattice bonds are broken with a probability that in non-isospin dependent percolation models is the only free parameter. The remaining connected clusters are identified with the fragments of the reaction, the bond-breaking probability with the excitation energy per nucleon [Con79].

In this chapter, the percolation model of Bauer et al. [Bau84] is modified by the explicit inclusion of isospin degrees of freedom, i.e., the lattice is comprised of protons and neutrons instead of just nucleons, in an attempt to reproduce the experimental data gained in the comparison of the multi-fragmentation in ^{112}Sn+^{112}Sn and ^{124}Sn+^{124}Sn collisions [Kun96]. We especially focus on the average number of intermediate mass fragments (IMFs, $3 \leq Z \leq 20$) $\langle N_{\text{IMF}} \rangle$ versus the number of charged particles N_c (Fig. 2.1, left panels), and versus the number of neutrons N_n (Fig. 2.1, right panels) detected. The full circles denote the results for the ^{124}Sn+^{124}Sn reaction, the circles the results for the ^{112}Sn+^{112}Sn reaction. The striking feature about these distributions is the "splitting" of $\langle N_{\text{IMF}} \rangle (N_c)$, and the position of the maxima in $\langle N_{\text{IMF}} \rangle (N_n)$. Both do not agree with common multi-fragmentation models, in which the distributions $\langle N_{\text{IMF}} \rangle (N_c)$ should lie on top of each other, and the positions of the maxima in $\langle N_{\text{IMF}} \rangle (N_n)$ should simply correspond to the ratio of neutron abundances in the respective isotopes.

Figure 2.1: Average number of intermediate mass fragments (IMFs) versus number of charged particles (left panels) and neutrons (right panels). The full circles denote the experimental results [Kun96] for a ^{124}Sn+^{124}Sn collision, the open circles the results for a ^{112}Sn+^{112}Sn collision, both at 40 MeV/A. The solid lines represent percolation simulation results for the heavier isotopes, the dashed lines for the lighter ones. The top row was calculated without any stability mechanism, in the second row, an evaporation mechanism is employed, in the third row a fission mechanism, the fourth row was achieved with a combination of both.

Figure 2.1


^{124}Sn
^{112}Sn
$\langle N_{\mathrm{IMF}} \rangle$
N_c
N_n
0
10
20
30
40
50
1
2
3
4
5

2.2 Implementation of the Isospin Dependent Percolation Model

For each simulated collision event, first the impact parameter is randomly selected. Then with a simple Monte Carlo integration, the number of protons and neutrons in the overlap zone of the two nuclei is determined. We employ an approximation in which the nucleons outside the overlap zone are neglected – we found this approximation to be appropriate by studying Boltzmann-Uehling-Uhlenbeck (BUU) simulations [Bau86] of the collisions at different impact parameters, which clearly showed distinct spectator regions in the final state even for small impact parameters. Figure 2.2 shows the outcome of the BUU simulations at different impact parameters. The simulations were run with $b = 1, 3$, and 5 fm, respectively, and the spectator regions were observed even 200 fm/c later.

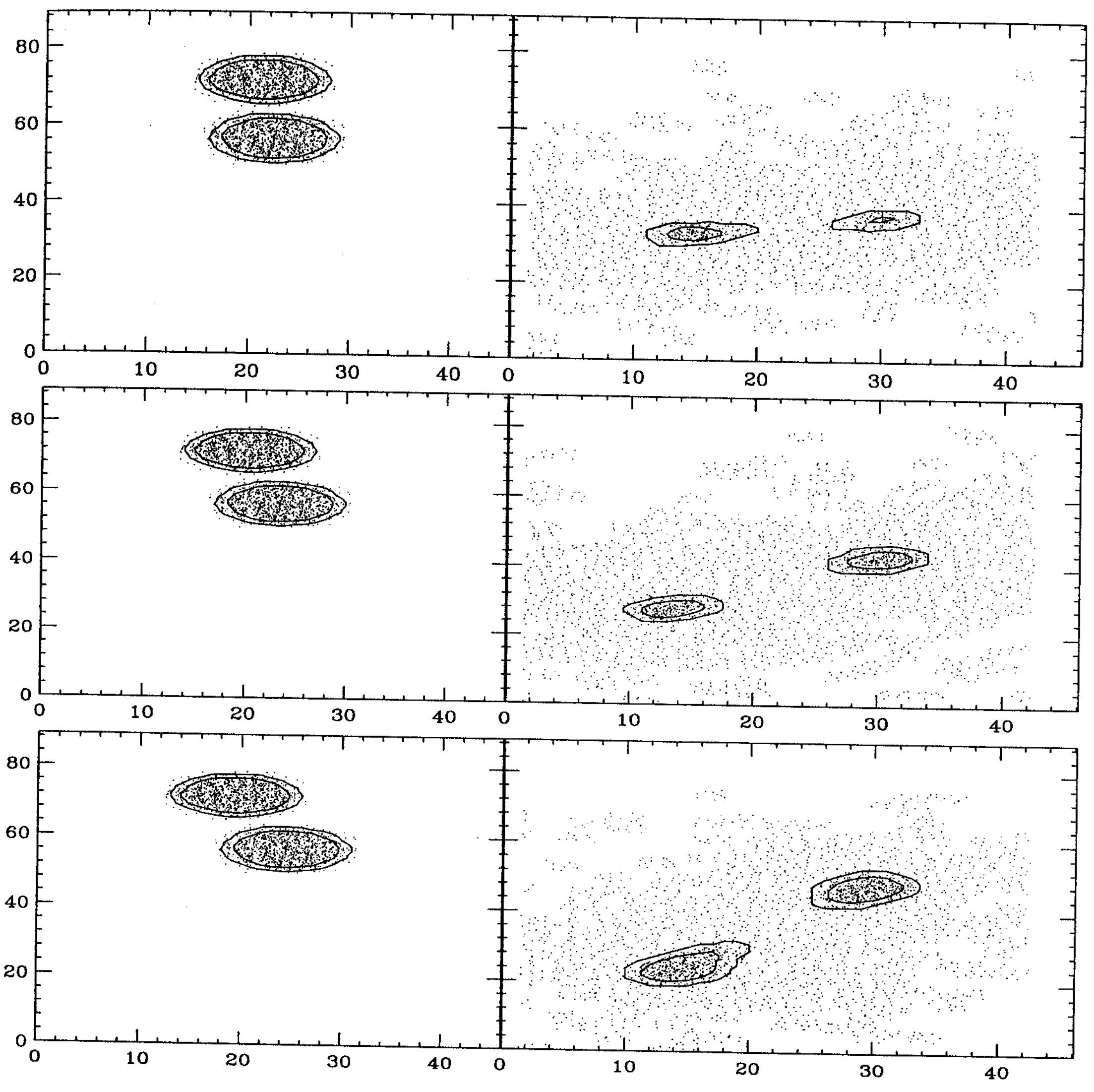


Figure 2.2: BUU simulations of the 40 MeV/A Sn-collisions at different impact parameters. The top plot was done with b = 1fm, the middle one with b = 3fm, and the bottom one with b = 5fm. The left plots show the initial configuration, the right plots the same scenario 200 fm/c later.

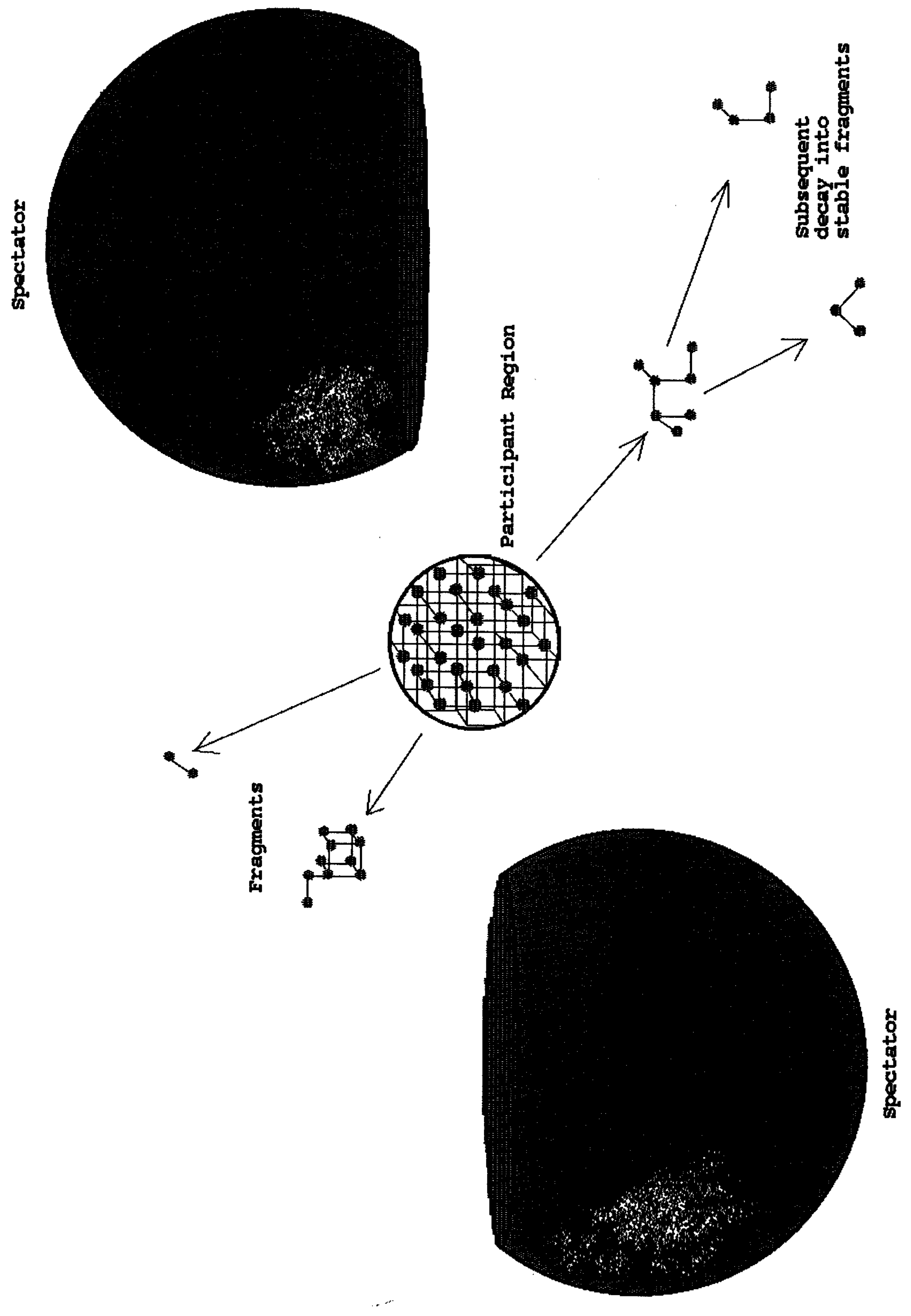


Figure 2.3: The nucleons in the participant (overlap) zone of the nuclei are distributed on a rectangular lattice. Bonds within the participant region are randomly broken, for the resulting fragments, different stability mechanisms are employed.

The nucleons in the overlap zone are randomly distributed on a rectangular lattice (see Figure 2.3).

The lattice bonds are then broken with a probability p, which we determined in two different ways: in one method we set it equal to a parameter p_0 which we obtain by fitting to the experimental data [Kun96], in the second method we choose p according to a Gauss distribution around p_0, i.e., for each simulation, p varies slightly in order to simulate excitation energy fluctuations. A comparison between both methods showed no significant difference in the outcome except for better statistics in the latter method for higher event multiplicities as large bond-breaking probabilities were included. Since we find the inclusion of excitation energy fluctuations to be more realistic, we settled for the latter method.

We then identify clusters of nucleons which are still connected with each other. However, since those clusters are not necessarily a stable configuration of protons and neutrons, we experimented with several different algorithms to achieve fragment stability (see Fig. 2.3). Methods included a re-distribution of protons and neutrons between the fragments, further fission of the fragments, evaporation of protons and neutrons from the fragments, and simulations with no additional stability criteria applied. Also, the definition of stability is not obvious: the experimental lifetime data applies to nuclei in their ground state and is not directly transferable to the fragments of a multi-fragmentation reaction. Since we found the outcome to only be slightly dependent on the definition used, we settled on a stability criterion where the fragments are required to have a ground state lifetime that is long enough for them to reach the detectors. As an unfortunate side effect of fission and evaporation mechanisms, however, the relationship between the bond breaking probability p_0 and the excitation energy is not obvious anymore, rather the combination of the initial bond-breaking and further mechanisms leads to an effective bond-breaking probability

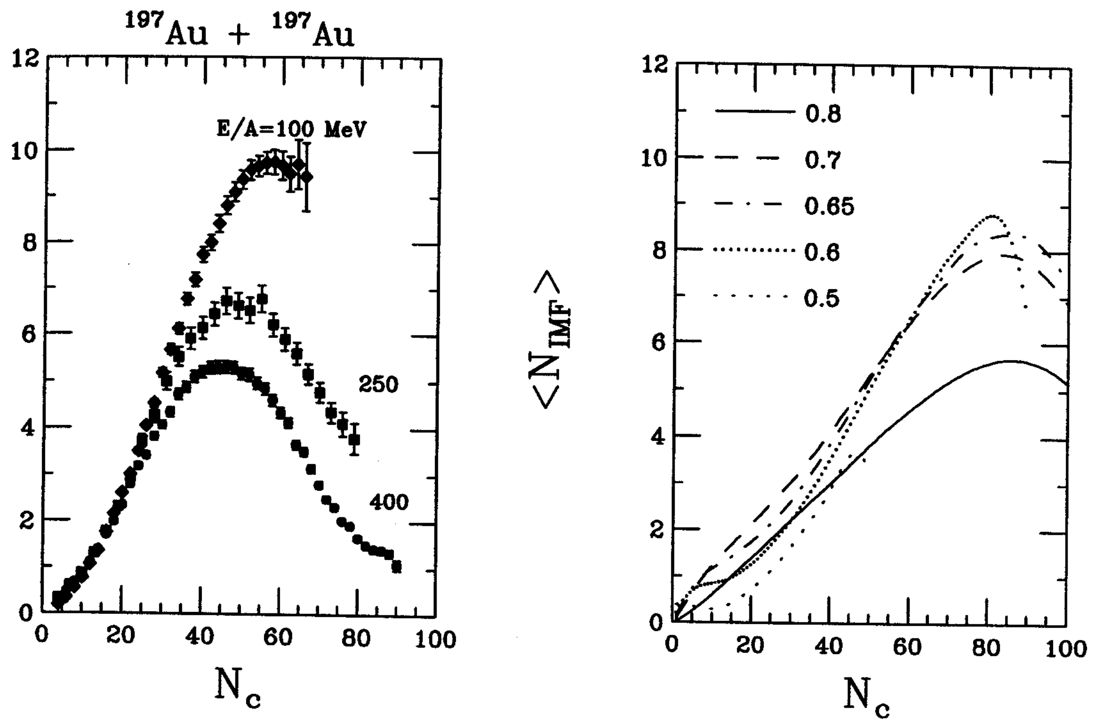


Figure 2.4: Experimental data [Tsa93] and percolation results for a ^{197}Au+^{197}Au collision. The experimental data is given for E/A=100, 250 and 400 MeV, the theoretical predictions refer to different bond-breaking probabilities p_0.

that is higher than p_0 — therefore, p_0 sets a lower limit for the excitation energy.

Of the order of $10^6 - 10^7$ events were simulated for each set-up; for each individual event the number of charged particles, neutrons and IMFs was recorded, where in accordance with the experimental data we employed detector efficiencies of 0.9 for all charged particles and 0.65 for neutrons.

2.3 Results and Conclusions

The code was first applied to the experimental results of Ref. [Tsa93]. Figure 2.4 shows the the average number of intermediate mass fragments ($3 \leq Z \leq 20$) versus the number of charged particles ($\langle N_{\text{IMF}}\rangle(N_c)$) for ^{197}Au+^{197}Au collisions. The experimental results in the left panel refer to different energies per nucleon [Tsa93], the curves in the right panel to percolation simulations at different bond-breaking probabilities p_0. It had been found earlier [Pha92] that percolation codes generally slightly underpredict the number of intermediate mass fragments, which was attributed to the possible existence of non-compact decay geometries. Overall, however, the model was found to reproduce the data reasonably well.

In the next step, we address the question of the reproducibility of the isospin-dependence found in Ref. [Kun96]. The top row of Fig. 2.1 shows the result of a simulation with a bond-breaking probability distribution centered around $p = 0.7$ and a half-width of 0.1. In this simulation, no stability mechanism is applied. The solid line corresponds to the simulation for the heavier isotope (experimental data indicated by full circles), the dashed line to the lighter isotope (open circles). The difference between the isotopes in the average number of IMFs versus the number of charged particles could not be reproduced, the outcome basically reflects the trivial autocorrelation that every IMF is a charged particle, the slope is determined by the ratio of IMFs versus lighter fragments. The linear relationship will eventually break down in events with high multiplicities when more and more fragments are smaller than IMFs. The experimental data shows that in the heavy isotope relatively fewer fragments with $Z < 3$ are formed, a trend that cannot be seen in the simulation. The difference in the average number of IMFs versus number of neutrons (right panel) can be reproduced, which, however, is not surprising: in the collision of the neutron-richer isotopes, more neutrons are emitted. The positions of the maxima, i.e., at 25 and 29 neutrons, respectively, correspond to the ratio of neutrons in the isotopes, 62 and 74, respectively ($25/29 \approx 0.86; 62/74 \approx 0.84$). In the experimental data the maxima are at 25 and 40 neutrons, the ratio of ≈ 0.63 is in-compatible with the simple explanation above. At high neutron numbers, statistics get rather unsatisfactory, and we do not reproduce the high neutron multiplicities seen in the experiment for the heavy isotope.

The second row of Fig. 2.1 shows the outcome of a simulation with an evaporation mechanism to achieve fragment stability: protons and neutrons are broken off the fragments until the remainder is stable. In the simulation shown, in case of $N > Z$, neutrons are broken off, and vice versa. In another simulation, protons and neutrons had been broken off randomly, which lead to slightly less IMFs. Again, differences in the IMF-distribution versus number of charged particles (left panel) could not be observed, however, for both isotopes, in comparison to the simulation without stability criterion,the ratio of IMFs to lighter particles decreased. In the IMF distribution versus number of neutrons (right panel), naturally higher neutron multiplicities are observed, the ratio of the maxima positions (32/40=0.8) remains compatible with the ratio of neutrons between the isotopes, though.

The third row of Fig. 2.1 results from a simulation a fission mechanism: an unstable fragment is broken into two fragments, if such a secondary fragment is unstable, it is again broken up into two fragments, and so on, until only stable fragments remain. The distribution of the secondary fragment sizes is chosen to be centered around 0.5 and falls off quadratically towards 0 at 0 and 1; a simulation with a flat distribution yielded similar results, it had only very slightly less IMFs. Both these mechanisms fit the IMF distribution versus number of neutrons rather well, the ratio of the positions of the maxima, $31/41 \approx 0.7$ is half-way between the expected 0.83 and the experimental 0.6, however, the different heights of the maxima could not be reproduced. The IMF-distribution versus charged particles again fails to show differences between the isotopes.

The forth row finally shows the result for a combination of the two mechanisms above, each unstable fragment undergoes evaporation or fission with equal probability. Apparently, this mechanism produces too many light fragments, both charged particles and neutrons. The ratio of the maxima in the IMF distribution versus number

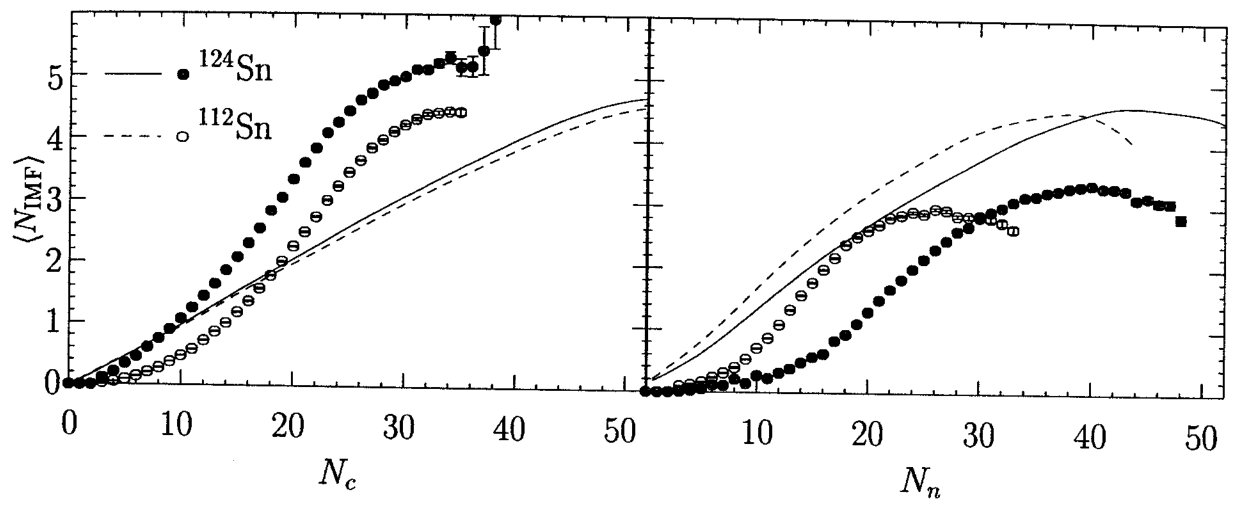


Figure 2.5: Results of a simulation with different bond-breaking probabilities for equal and different nucleon bindings ($p_{0d} = 0.6$; $p_{0e} = 0.9$).

of neutrons is 33/44=0.75.

In deviation from the established percolation models, we also worked with different break-up probabilities for bonds between protons and protons, neutrons and neutrons, and protons and neutrons, that is, a probability p_{0e} for equal pairings, and a probability p_{0d} for unequal pairings, $p_{0e} \geq p_{0d}$. As it turns out, in this method the outcome depends significantly on the distribution of protons and neutrons on the lattice: in the case of a highly ordered configuration where except for the excess neutrons both types of nucleons are distributed in an alternating way ("salt crystal"), i.e., in general every proton has six neutrons as nearest neighbors and vice versa, the excess neutrons being put in randomly as "impurities," one notes significant differences between both Sn-isotopes in the distribution of intermediate mass fragments versus charged particles – however, these are still much smaller than in the experiment, as Fig. 2.5 illustrates this for $p_{0e} = 0.9$ and $p_{0d} = 0.6$. Even though this effect leads to a slightly closer resemblance of the experimental data, we consider it to be an artifact since it nearly completely vanishes with a purely random distribution of protons and neutrons on the grid. We attribute this effect to the fact that in a pure "salt-crystal" configuration every bond has the break-up probability p_{0d}; every impurity will in general lead to the introduction of six bonds with break-up probability p_{0e}, and therefore has a large impact. As a result, disregarding surface effects, in the collision between the lighter isotopes about 79% of the bonds are of type p_{0d} and 21% of type p_{0e}, while for the heavier isotopes the percentages are 61 and 39, respectively. In the random configuration, however, there is no such amplification, the percentages are 58 versus 42 for the lighter isotope, and 56 versus 44 for the heavier isotope.

Also, simulations were run with a "neutron skin," which influenced the ratio of protons and neutrons in the overlap zone for different impact parameters: for small impact parameters the ratio of neutrons to protons was higher than for large impact parameters, the size of the neutrons skin were determined by a Hartree-Fock calculation[Bro96]. However, also these simulations could not improve the agreement with experimental data.

In conclusion, the experimental results could only be reproduced in part. The main discrepancies are:

- The difference between the two isotopes in the average number of IMFs versus number of charged particles could not be reproduced.
- The difference in the maximum values of the IMF distributions versus number of neutrons could not be reproduced.
- The experimental positions of the maximum number of IMFs versus number of neutrons is in-compatible with the trivial shifting due to higher neutron abundance that is found in the simulation.

The discrepancies found between the data and this basically geometrical approach indicate that effects outside of percolation theory are important. The nuclear structure of the fragments as well as sequential feeding might play a role. Most important however seems the role of preequilibrium emission which may not only effect the sorting axis but as well determines the N/Z composition of the fragmenting system.

Chapter 3

Nuclear Flow in Consistent Boltzmann Algorithm Models

3.1 Introduction

Heavy-ion collisions are of great importance in answering the question how nuclear matter changes its properties under different temperatures and pressures.[Cze86] When two large nuclei collide, the nuclear matter is compressed, and due to two-body collisions between nuclear constituents also the temperature rises to about $50 - 100$ MeV (about 5 to 10 times 10^{11} Kelvin). Unlike for macroscopic matter, for nuclear matter possible phase transitions etc. are unfortunately not directly observable: both the length (about 10 fm$=10^{-14}$ m) and the time scale (about 60 fm$/c = 2 \times 10^{-22}$ seconds) of nuclear collisions prohibit probing the nuclear matter while it is still at that high temperature and density. Instead, the nuclear matter has already cooled down, expanded, and formed more stable configurations before it hits the detectors; every observation can only be indirect.

However, these late reaction products still carry information that helps to reconstruct what happened in the early stages of the collision. For example, the collective flow of the reaction products contains information about the compressibility of nuclear matter. With the exception of the merely theoretical case of a head-on collision of the two nuclei (vanishing impact parameter), from straightforward considerations it is expected that in the collision nuclear matter of the two partners is pushed outwards perpendicular to the beam within the plane defined by the impact parameter. The harder in the sense of compressibility the nuclear matter would be, the stronger this effect would be expected to be. Also, from this very simple consideration one would expect the flow to always increase with the beam energy. However, since the interaction between nucleons in addition to a short range repulsion has a long range attraction, this simple picture does not hold true: at low energies, the long range part of the nuclear interaction dominates, the nucleons are attracted, and flow is generated by the nucleons being scattered *towards* each other. At energies around 70-140 MeV/nucleon depending on the mass of the reaction partners both effects cancel out each other, the flow disappears in the experiment; simulations to-date tend to underpredict that value.[Wes86] At high energies, the nucleons collide at short range, and the mechanism for flow indeed corresponds to the simple consideration above.

To extract quantitative information about and gain insight into the microscopic processes within nuclear matter, however, it is necessary to have a dynamical model to simulate the collisions themselves. Comparision between the predictions for the final stage gained by the simulation on the one hand, and the actual experimental results on the other hand are used to refine the model for the early stages. The simulation of heavy-ion collisions unfortunately is equivalent to solving a quantum-mechanical many-body problem, which to date is not fully possible. Different approaches have been made to nevertheless approximate the solution. Early approaches were based on hydrodynamics, they used a nuclear equation of state, but the mean free path of the nucleons was assumed to be so small that the nuclei in the collision merely resembled splashing droplets. At the other extreme, there were models that due to their extremely long mean free path basically resembled colliding gas clouds.[Das93]

To model the situation more realistically, testparticle based models were developed. Here, a nucleon and other nuclear constituents are represented by a number of testparticles, their individual trajectory is followed rather than being concerned only with the global properties of some nuclear "fluid" or "gas." Obviously, these microscopic methods, which can be implemented in different ways, have the general property of being computationally more intense than the macroscopic ones.

One method is that of Molecular Dynamics,[Aic91] in this method both the long-range attractive (soft) and the short-range repulsive (hard) part of the particle interaction is parametrized by potentials, the trajectories of the testparticles are continously updated in response to the local potential.

Another semi-classical particle-based method is the Boltzmann-Uehling-Uhlenbeck (BUU) approach. Here the soft part of the interaction is represented by mean fields, while the hard part is given by an explicit collision term. The collision term itself can again be represented in different ways, especially the criteria for a collision to happen are model dependent. In many codes, this decision is based on geometrical considerations, for example, a collision is generated at the point of closest approach between two particles. The BUU method is being used by various groups.[Ber88, Aic96, Bau86]

One particular implementation of the collision term in BUU codes is the Direct Simulation Monte Carlo approach (DSMC), see for example Lang et al.,[Lan93] and Danielewicz.[Dan95] In this approach, collisions between the testparticles are not generated through geometrical and particle-trajectory based criteria, but stochastically in a way that the correct collision rate is reproduced. In a collision only momenta and energy of the particles are changed, while the particles themselves stay in place until the next advection step – the particles are assumed to be pointlike.

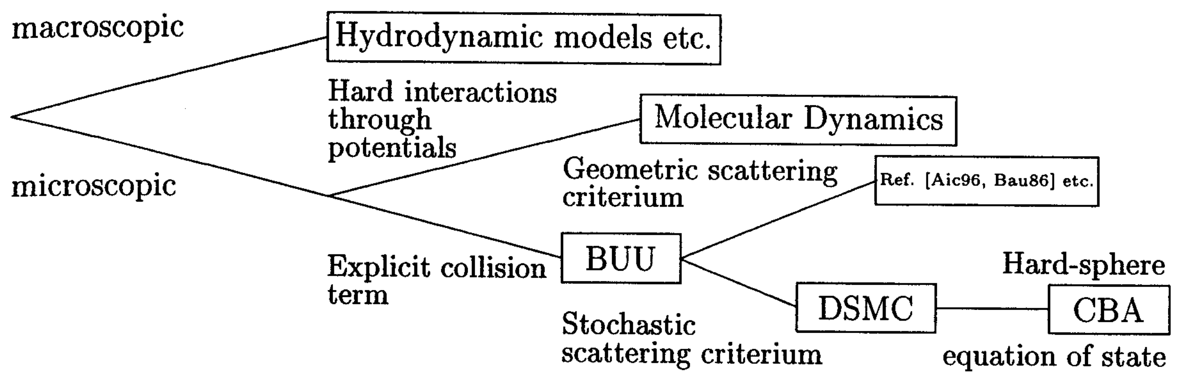


Figure 3.1: Overview of techniques to simulate heavy-ion collisions.

It was suggested that in order to reproduce a Hard-Sphere Boltzmann Equation, the DSMC approach should be extended by an additional advection that should take place after any collision,[Ale95] and by a modification the collision probability itself[Ale95, Res77] (Enskog Theory).

Figure 3.1 summarizes these different techniques.

This advection is supposed to push the collision partners away from each other according to their hard-sphere radius, and the collision probability is adjusted to take into account the excluded volume of the hard spheres and screening and shadowing effects. The modified DSMC method is called Consistent Boltzmann Algorithm (CBA). The modifications ensure a non-vanishing second virial and change the equation of state for the scattering process from that of an ideal gas to that of a hard-sphere gas; their effect on the calculated value of directed nuclear collective flow in heavy ion collisions is analysed, and it is found that the flow slightly increases.

3.2 Theoretical Background

In the DSMC approach, the positions and momenta of the particles are evolved in a two-step process, namely advection and collisions, corresponding to one timestep of the simulation. During the advection step the particles are propagated according to their momenta. During the collision step first the particles are sorted into spatial cells of volume V. Then out of the n particles within a given box, at random, m combinations are chosen and scattered with the probability

$$W = \frac{\sigma(\sqrt{s})v_{\text{rel}}\Delta t}{NV}\frac{n(n-1)/2}{m}\,, \tag{3.1}$$

where $\sigma(\sqrt{s})$ is the energy-dependent elementary hadron-hadron cross section, N is the number of testparticles representing one nucleon in a full-ensemble testparticle algorithm [Wel89], Δt is the timestep length, and v_{rel} is the relative velocity of the particle pair [Lan93]. In the limit $V \to 0$, $\Delta t \to 0$, $N \to \infty$, the solutions of this method have been shown to converge to the exact solution of the Boltzmann equation (without mean field contribution)[Bad89],

$$\partial_t f_1 + \boldsymbol{v} \cdot \frac{\partial f_1}{\partial \boldsymbol{r}} = (\partial_t f_1)_{\text{coll}} \;. \tag{3.2}$$

Here, $f_1(\boldsymbol{r}, \boldsymbol{v}; t)$ is the *one particle distribution function* , which, according to Boltzmann, is defined in such a way that

$$f_1(\boldsymbol{r}, \boldsymbol{v}; t)\; d\boldsymbol{r}\; d\boldsymbol{v}$$

is the number of particles that at a time t are within a volume element $d\boldsymbol{r}$ around $\boldsymbol{r}$ and have a velocity $d\boldsymbol{v}$ around $\boldsymbol{v}$. The infinitisimal volume elements have to be taken with a grain of salt, since taken literally, within a *mathematically* infinitisimal volume element, f_1 could either be '1' or '0' – the pointlike particle could either be inside the infinitisimal volume, or not. Rather, the volume element has to be a *physical* element

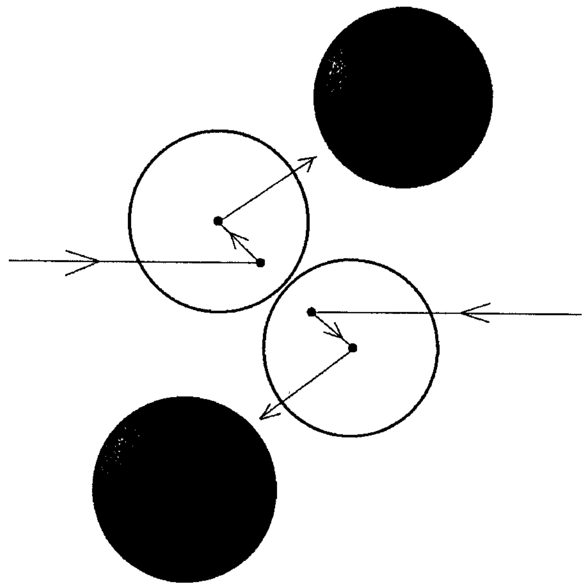


Figure 3.2: Additional displacement of scattering partners due to their finite volume

that is *microscopically* infinite – small enough so that physical properties do not vary appreciably over its extension, but big enough to contain a large number of particles.

This approach does not take into account the finite size of the nucleons; the testparticles are point-like, and if it was not for the contribution of the mean field, they would be following an ideal gas equation of state. It has therefore been suggested by Alexander et al. [Ale95] to include an extra displacement $\boldsymbol{d}$ of the collisions partners,

$$\boldsymbol{d} = \frac{1}{2}\frac{\boldsymbol{v}_r' - \boldsymbol{v}_r}{|\boldsymbol{v}_r' - \boldsymbol{v}_r|}\sqrt{\frac{\sigma(\sqrt{s})}{\pi}} \ , \tag{3.3}$$

$\boldsymbol{v}_r = \boldsymbol{v}_1 - \boldsymbol{v}_2$ being the velocity difference before, and $\boldsymbol{v}_r' = \boldsymbol{v}_1' - \boldsymbol{v}_2'$ being the velocity difference after the collision. Particle 1 is displaced by $\boldsymbol{d}$ and particle 2 by $-\boldsymbol{d}$. This additional advection pushes the nucleons apart according to their hard-sphere radius. Figure 3.2 illustrates this mechanism.

Within the Boltzmann equation, this displacement corresponds to change of the collision term

$$(\partial_t f_{1,2})_{\text{coll}} = \int d\boldsymbol{v}_1 \int d\boldsymbol{v}_2' \int d\boldsymbol{v}_1' W(\boldsymbol{v}_2, \boldsymbol{v}_1; \boldsymbol{v}_2', \boldsymbol{v}_1') \left[f_{1,2}' f_{1,1}' - f_{1,2} f_{1,1}\right] \ , \tag{3.4}$$

W again being the scattering probability, and the f's being the respective one-particle distribution functions before and after the scattering for the two particles, i.e.,

$$f_{1,1} = f_1(\boldsymbol{r}, \boldsymbol{v}_1; t), \quad f'_{1,1} = f_1(\boldsymbol{r}, \boldsymbol{v}'_1; t) \quad \text{for Particle 1,} \tag{3.5}$$

$$\text{and } f_{1,2} = f_1(\boldsymbol{r}, \boldsymbol{v}_2; t), \quad f'_{1,2} = f_1(\boldsymbol{r}, \boldsymbol{v}'_2; t) \quad \text{for Particle 2.} \tag{3.6}$$

With the additional displacement, it is

$$f_1(\boldsymbol{r}, \boldsymbol{v}_2; t)\, f_1(\boldsymbol{r}, \boldsymbol{v}_1) \longrightarrow f_1(\boldsymbol{r} + \boldsymbol{d}, \boldsymbol{v}_2; t)\, f_1(\boldsymbol{r} - \boldsymbol{d}, \boldsymbol{v}_1; t) \ . \tag{3.7}$$

The same replacement has to be done with $\boldsymbol{d} \to -\boldsymbol{d}$ for the inverse term $f'_{1,2} f'_{1,1}$.

It is not obvious right away how this displacement scales with N. However, as in the mean free path $1/((\sigma/N)(N\varrho))$ of a testparticle, ϱ being the nuclear density, there is no N dependence, the average number of collisions that a certain testparticle is involved in is independent of N. Therefore, in order to achieve the same total displacement during the course of the simulation, the individual displacement per collision should not depend on N either. The testparticles are therefore pushed apart according to the nucleonic radius, and not according to the effective testparticle radius.

The finite radius of the particles also makes it impossible for one particle to be within the "hard core" of the others, and thereby from the available volume V a fraction

$$\frac{n}{N} \cdot \frac{4}{3}\pi a^3 = V\varrho \cdot \frac{4}{3}\pi a^3 \tag{3.8}$$

is occupied, where a is the average "hard core radius" of the nucleon, and n/N is the number of nucleons in the respective box. If one only looks at the centers of the nucleons, they cannot be closer together than $2a$ – in that respect, another view of the scenario would be to assign each "point nucleon" a "sphere of influence" with radius $2a$. We obtain a by randomly picking n particle combinations for a respective box and calculating their cross sections $\sigma_i(\sqrt{s_i})$.

The Pauli principle is approximately taken into account in the following way: since collisions between particles from the same nucleus that had never scattered before are assumed to be Pauli-forbidden, in the determination of the average cross section only particle combinations with partners that are not from the same nucleus, unless at least one of them had scattered before, are taken into account. According to the Golden Rule, the cross section is proportional to the *available* phase space, however, since the particle combinations chosen for the determination of the average cross section were chosen without any further restrictions, a momentum space volume of $4/3\pi(p_B+p_F)^3$, where p_B is the momentum of the beam per nucleon, and p_F is the Fermi momentum, was implicitly assumed. In reality, the two nuclei occupy part of the momentum space, namely, $2\cdot 4/3\pi p_F^3$. Thus, we adopt the following approximation for the effective radius a, i.e.,

$$a = \frac{1}{2}\sqrt{\frac{1}{\pi}\left(\frac{1}{n}\sum_{i=1}^{n}\sigma_i(\sqrt{s_i})\right)\cdot\left(1-2\frac{p_F^3}{(p_F+p_B)^3}\right)}\,. \tag{3.9}$$

For small E_{lab}, this effective radius is about 0.84 fm, in the range between 100-400 MeV it is about 0.47 fm. This is slightly larger than what has recently been suggested in Ref. [Dan96], there, the effective radius derived from delays in elementary processes is about 0.6-0.8 fm and 0.15-0.3 fm, respectively.

Due to the reduced volume

$$\widetilde{V} = \left(1-\varrho\cdot\frac{4}{3}\pi a^3\right)V \tag{3.10}$$

alone, a modified higher scattering probability

$$\widetilde{W} = \frac{V}{\widetilde{V}}W \tag{3.11}$$

has to be used – the particle have less free space between them and therefore scatter more often, see Figure 3.3.

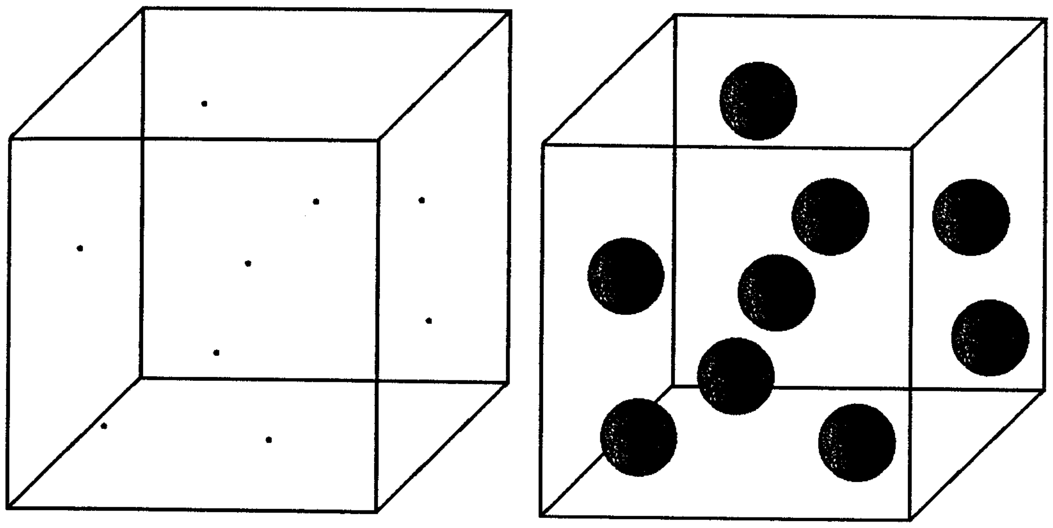


Figure 3.3: Reduced volume due to finite particle radius. In previous implementations, the testparticles were pointlike (left), which left more free space between the particle. The current implementation (right) takes into account the volume taken up by the testparticles.

However, the scattering probability is lowered again by another effect: the particles are screening each other. A particle might not be available for scattering with another particle because there might be a third particle in between, an effect that is illustrated in Figure 3.4.

Through this effect, a portion ΔS of the total 2-dimensional surface projection $S = \pi(2a)^2 = 4\pi a^2$ of the "spheres of influence" of the particles is not available for scattering, which lowers the collision frequency by a factor $\left(1 - \overline{\Delta S}/S\right)$, where $\overline{\Delta S}$ is the average screened surface area. With x being the distance of the center of spheres 1 and 3 in Fig. 3.4. Allowing the distance to vary, within a range $(x, x + dx)$, the average number of "No. 3 - spheres" is $4\pi\varrho x^2 dx$, each of them screening a surface ring area $2\pi a(a - x/2)$ of sphere 1. Therefore, the average screened area is

$$\overline{\Delta S} = \int_a^{2a} dx\; 4\pi\varrho x^2 \cdot 2\pi a \left(a - \frac{x}{2}\right) = \frac{11\pi^2 \varrho a^5}{3} \,. \tag{3.12}$$

Recalling the change in collision frequency, this effect leads to a reduction of the

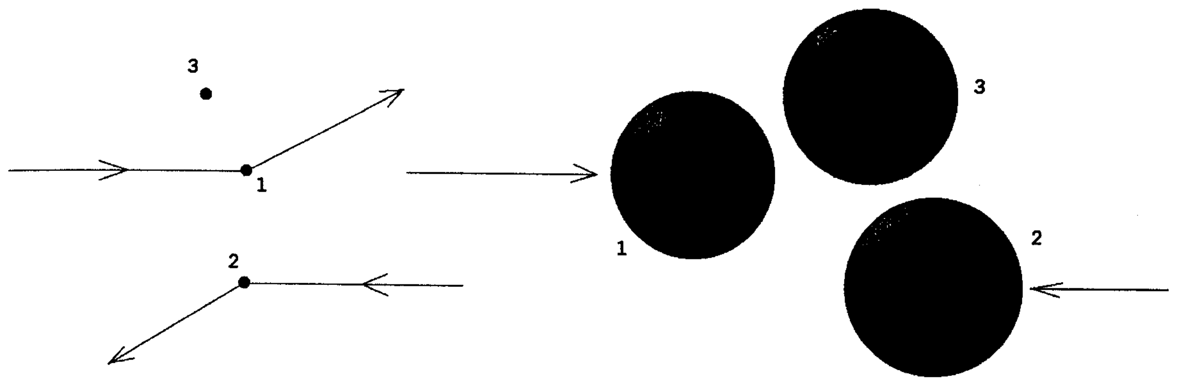


Figure 3.4: Screening effect due to the finite volume of the testparticles. On the left, the pointlike particles 1 and 2 could scatter with each other, particle 3 is not “in the way.” On the right, due to the finite volume, they are prevented from scattering, particle 1 would scatter with particle 3 first.

scattering probability by a factor of

$$\left(1-\frac{\overline{\Delta S}}{S}\right)=\left(1-\varrho\cdot\frac{11}{12}\pi a^3\right)\ . \tag{3.13}$$

Again, the product $\varrho \cdot a^3$ is independent of the number of testparticles per nucleon N. Including this factor, the modified scattering probability is

$$W' = Y^E W\ , \tag{3.14}$$

where

$$Y^E=\frac{1-\varrho\cdot\frac{11}{12}\pi a^3}{1-\varrho\cdot\frac{4}{3}\pi a^3}=\frac{1-11b^E\varrho/8}{1-2b^E\varrho}\ ,\quad b^E=\frac{2}{3}\pi a^3\ , \tag{3.15}$$

b^E being the second virial coefficient as yielded by the Enskog Theory of the dense hard-spheres fluid [Res77]; see figure 3.5.

The implementation of the modifications makes the second virial of the hard part of the interaction non-vanishing; b^E is positive and therefore leads to an increase in preasure. This is partly compensated by the negative virial b^S that is due to the soft (mean field) part of the interaction; see for example [Dan96]. The equation of state

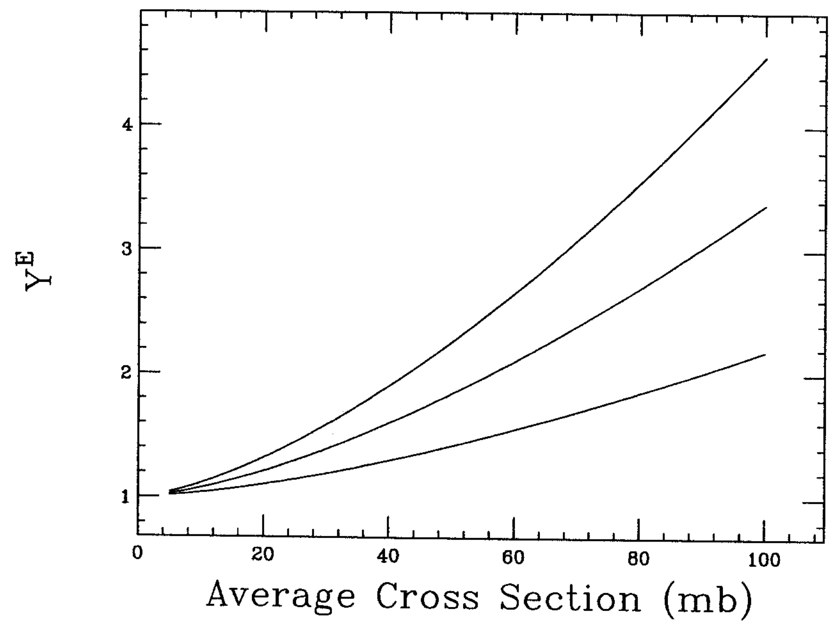


Figure 3.5: The scattering enhancement factor Y^E as a function of the average cross section. Shown is Y^E for $\varrho = 1, 2, 3\varrho_0$.

deviates from that of an ideal gas by both contributions, i.e.,

$$P = \varrho kT \left(1 + \varrho(b^E + b^S) + \ldots\right) . \tag{3.16}$$

One should note at this point that the Enskog Theory is non-relativistic; both the advection vector $\boldsymbol{d}$ and the excluded volume, therefore also Y^E, are calculated in a frame-dependent way.

3.3 Results and Conclusion

Our numerical calculation is based on the MSU BUU-code by Bauer et al. [Bau86] which was modified from a geometrical formulation of the collision term to a stochastic formulation according to Ref. [Lan93]. Only NN collisions were taken into account, which for the energies considered turned out to be a justified approximation. A full-ensemble and a parallel-ensemble implementation proved to have similar results; results for different systems were compared with both Refs. [Bau86] and Ref. [Dan95]; they were found in satisfactory agreement. An interesting side-result at this stage however was that changing the algorithm from geometrical to stochastic scattering increased the frame-of-reference dependence of the result: when running the simulation within the lab-frame we found an asymmetry in the flow which we attribute to the fact that in this frame the projectile is Lorentz-contracted and the target is not. Therefore within a spatial box inside of the overlap-zone of the two nuclei there are many more testparticles originating from the projectile than from the target, resulting in an asymmetry of the respective scattering rates. We are currently trying to overcome this problem by γ-dependent modifications of the scattering probabilities, however, so far with only little success. The flow-asymmetry vanishes when the calculation is performed in the c.m.-frame, which is what we did in this work. The geometry-based code did not appear to be sensitive to this asymmetry, however, Lorentz invariance certainly still is an issue, see Chapter 4. Finally the full-ensemble version of the stochastic code was modified according to Ref. [Ale95], and the scaling of the collision rate and the flow with the number of testparticles N was checked.

We simulated an (Au,Au)-collision at projectile energies of 250 and 400 MeV per nucleon, and $b = 3$ fm over a total time of 70 fm/c. These energies are large enough so that the repulsive (hard) part of the interaction dominates, but, as already pointed out, low enough so that inelastic NN scattering can still be neglected. The timestep length was 0.1 fm/c, the volume V was approximately 1.8 fm^3, the number of testparticles per nucleon N was 250, m in equation (3.1) was chosen to be n^2, and we used a soft momentum-dependent equation of state. Figure 3.6 shows the evolution of the configuration in the reaction plane.

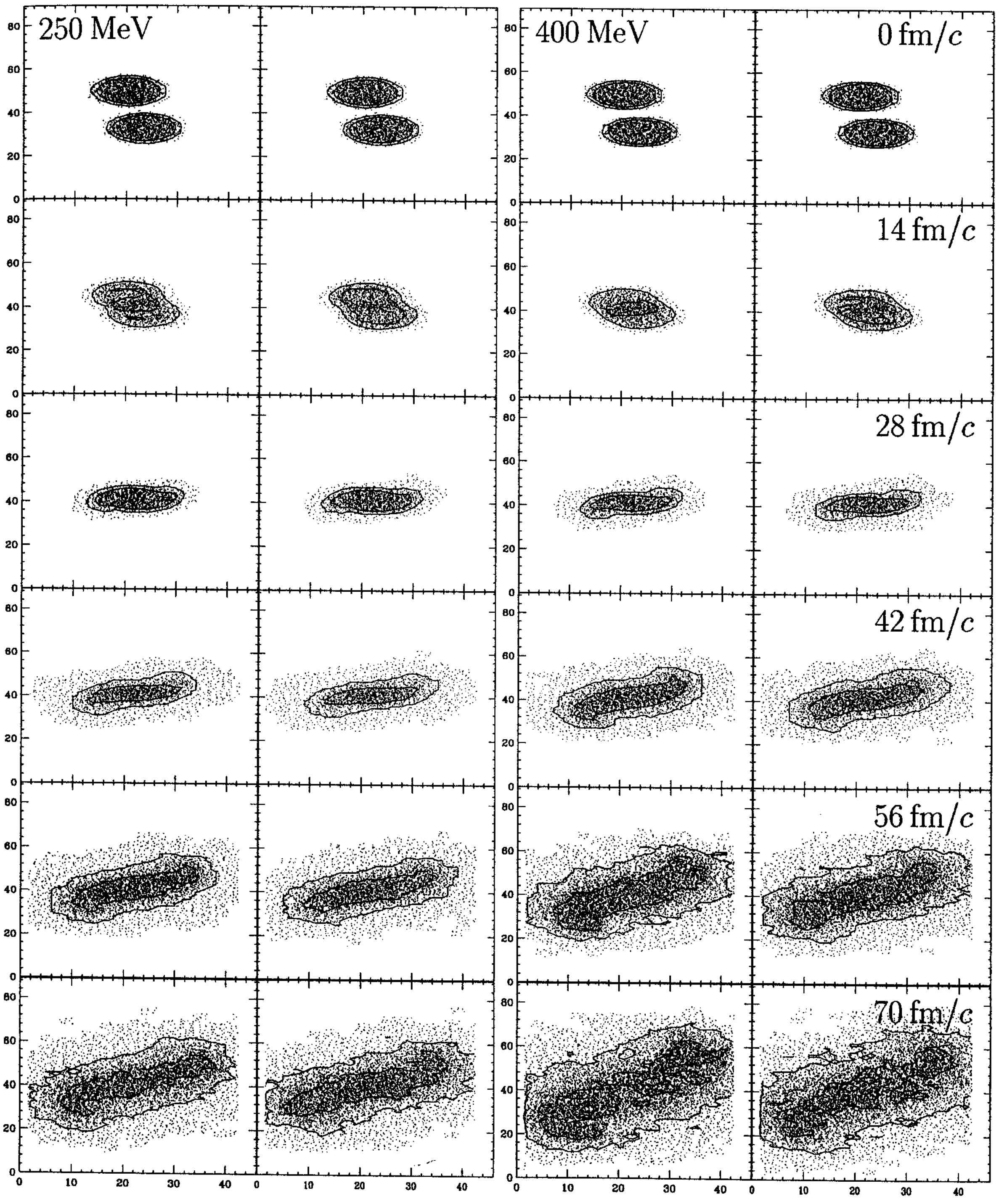


Figure 3.6: The nucleon configuration in the reaction plane at different times during the collision, shown is the nuclear density $\varrho(x, y = 0, z)$.

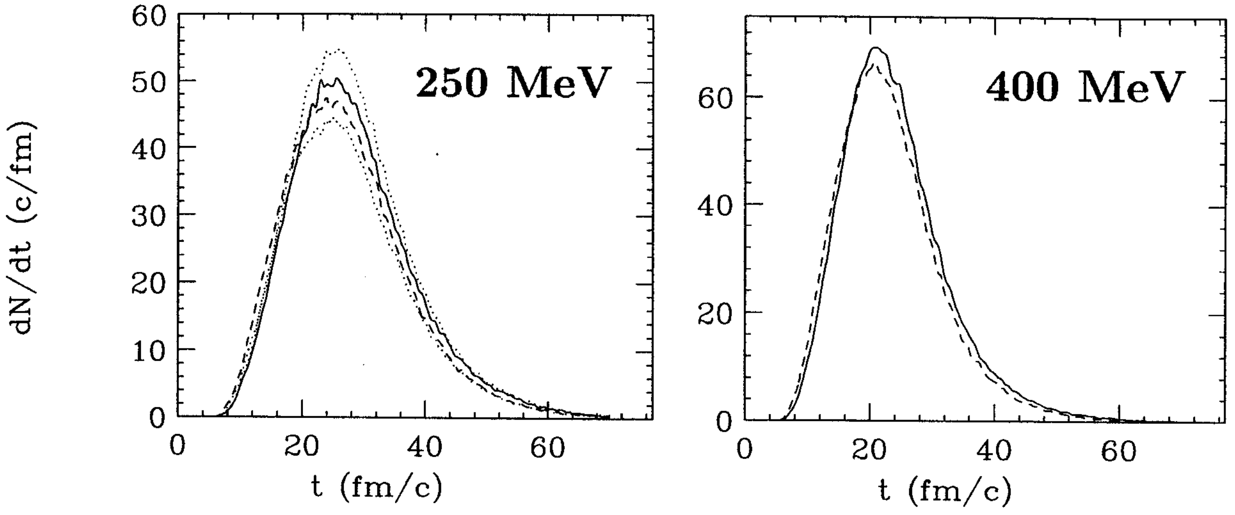


Figure 3.7: Collision rate for a 250 and 400 MeV (Au,Au) collision with $b = 3$ fm versus time.

The result for the unmodified algorithm is shown on the left, the result for the modified algorithm on the right, respectively. As expected, the nuclei disintegrate slightly more violently due to the additional advection, resulting in a lower nuclear density. A calculation with the additional advection alone revealed that therefore also the collision rate decreases. However, this is partly compensated by the modified scattering probability Eq. (3.15). On the other hand, a calculation with the modified scattering probability alone shows a strongly enhanced collision rate, as expected. The left panel of figure 3.7 illustrates this for the 250 MeV collision, the right panel shows the collision rates for the 400 MeV collision. The solid curve refers to the unmodified simulation, the dashed curve to the modified one. For the 250 MeV collision two dotted lines were added, the upper line refers to a calculation where only the scattering enhancement was taken into account, the lower line to a calculation that only incorporated the additional advection.

Figure 3.8 shows the average final transverse momentum versus the reduced rapid-

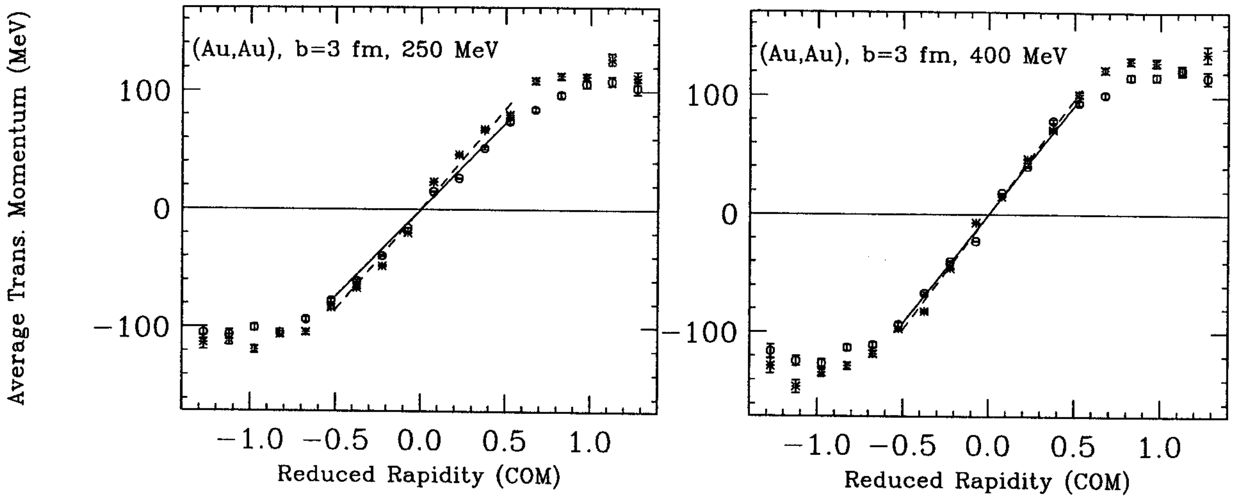


Figure 3.8: Average final transverse momentum versus reduced rapidity of the protons in a 250 MeV (left panel) and 400 MeV (right panel) (Au,Au) collision with $b = 3$ fm.

ity as an indicator for nuclear flow. The reduced rapidity is the rapidity $Y_{\rm cm}$ devided by the rapidity of the beam, which for the 250 MeV collision is approximately 0.36, and for the 400 MeV collision approximately 0.45. From the slope of a linear fit around the origin, one can determine the nuclear flow. The circles and the solid fit refer to the unmodified, the stars and the dashed fit to the modified algorithm.

For the 250 MeV collision, at $b = 3$ fm the flow is $\approx$ 147 MeV/(c·Unit of Red. Rap.) for the unmodified, and $\approx$ 170 MeV/(c·Unit of Red. Rap.) for the modified algorithm. A calculation that only took into account the scattering enhancement resulted in $\approx$ 166 MeV/(c·Unit of Red. Rap.), a calculation that only incorporated the additional advection in $\approx$ 159 MeV/(c·Unit of Red. Rap.). For the 400 MeV collision, at $b = 3$ fm the flow for the unmodified algorithm is $\approx$ 185 MeV/(c·Unit of Red. Rap.); for the modified algorithm it is $\approx$ 195 MeV/(c·Unit of Red. Rap.).

One concludes that in this specific simulation the flow increases by approximately 16% for the 250 MeV collision, and 5% for the 400 MeV collision, with the introduction of the new algorithm. Introducing only the additional advection, as already pointed

Table 3.1: Impact Parameter Averaged Analysis ($b \leq 5$ fm) and Experimental Data

Energy	Nuclear Flow			
	Unmodified	Modified	Plastic Ball[Gus88]	EOS[Par95]
(MeV)	(MeV/(c·Unit of Reduced Rapidity))			
250	132	148	130	119
400	166	185	169	151

out, the collision rate slightly decreases, however, the nuclear flow increases by about 8% for the 250 MeV collision. With the introduction of the modified scattering probability alone, the flow increases by about 13%. The fact that the contributions from both modifications do not "add up" indicates that a perturbative approach to their representation would not be justified.

An impact parameter averaged analysis for 250 MeV collisions with $b \leq 5$ fm resulted in approximately 132 MeV/(c·Unit of Reduced Rapidity) nuclear flow for the unmodified, and 148 MeV/(c·Unit of Reduced Rapidity) for the modified algorithm (12% increase). The Plastic Ball data indicate approximately 130 MeV/(c·Unit of Red. Rap.) [Gus88] nuclear flow, the EOS data only 119 MeV/(c·Unit of Red. Rap.) [Par95]. For 400 MeV collisions the same calculations resulted in in approximately 166 MeV/(c·Unit of Red. Rap.) nuclear flow for the unmodified, and 185 MeV/(c·Unit of Red. Rap.) for the modified algorithm (11% increase; Plastic Ball: $\approx$ 169 MeV/ (c·Unit of Red. Rap.) [Gus88]; EOS: $\approx$ 151 MeV/(c·Unit of Red. Rap.) [Par95]). Table 3.1 summarizes these results.

Overall we found the effect of the additional advection and the modified scattering probability to be significant, but not crucial. Their implementation moves the outcome of the simulations away from the experimental results. This indicates the need for an in-medium reduction of the NN cross section. This type of reduction was first found to be needed in studies of the disappearance of flow [Wes86] and later also in theoretical studies based on thermodynamic T-matrix theory at finite temperature [Alm95]. These results were obtained by algorithm with closest approach techniques. If one wishes to address the question of the nuclear equation of state with a DSMC algorithm, however, the corrections discussed in the present paper should be taken into account.

3.4 Outlook

In this chapter, the virial corrections to heavy ion collisions in analogy with the theory of gases have been studied numerically.

Recently, Morawetz [Mor97] introduced a more complete set of effective shifts that represent mean values of various non-localities of the scattering integral. These shifts enter the scattering integral in a form known from the theory of gases, however, the set of shifts is larger than the one intuitively expected. "Particle diameters" and other non-localities of the scattering integral are given in form of derivatives of the phase shift in binary collisions. It can be shown that the shifts reduce to a "particle diameter" d for the classical limit of hard spheres collisions. In the case of nuclear matter, the three distinguished space shifts maintain.

The derived nonlocal shifts are possible to be interpreted within a classical picture of scattering:

- As already indicated in this chapter, one can imagine that a hard sphere shift Δ_d has to be included in order to simulate hard sphere behaviour of the colliding particles, because the two particles cannot approach each other nearer than the hard core distance Δ_{d}.

- Next one has to account for the delay of asymptotic scattering with respect to the real trajectory of two colliding particles. Danielewicz and Pratt have recently discussed these time delays for equilibrium processes [Dan96]. This leads us to the picture of two particle sticking together and moving as a molecule with their center of mass velocity for a time Δ_t.
- The two particles are forming a molecule which can rotate during their correlated motion. This gives rise to an additional rotational shift Δ_ϕ.

An extension of the above approach to incorporate the additional shifts is desirable.

out much of the collision hard processes dominate which are described by elementary interactions between quarks, antiquarks and gluons (partons) as essentially semiclassical particles. After parton initialization according to some reasonably chosen nucleon structure functions, spacetime propagation is accomplished by discretizing time into units Δt_{step} and updating phase space densities according to relativistic transport equations including a crucial collision term.

Scattering processes in this theoretical framework are assumed to be non-retarded which, on a microscopic level, leads to information transport with velocities that can approach $\sqrt{\frac{\sigma}{\pi}}/\Delta t_{\text{step}}$, where σ is the parton-parton cross section. That is, it can increase without reasonable bound. As energies for the individual interactions decrease the corresponding cross sections increase. The timestep Δt_{step} is for reasons of convergence chosen to be less than the parton mean free paths divided by their velocities: of the order a few thousandths of a fm/c. Two problems occur immediately: On a macroscopic level a series of subsequent causality violating interactions can lead to shock-waves propagating faster than the speed of light. This is clearly unphysical. On a microscopic level the time-ordering of the incoming and outgoing partons of a scattering process becomes frame-dependent and Lorentz covariance is lost – this also is unphysical. These two problems are especially serious in ultrarelativistic parton cascades since collision rates are high, and the mean free paths approach the inter-particle distance. Problems arise from describing quantum-dynamical processes in a semi-classical picture, and from the demand of Lorentz-invariance in an equal-time-character simulation. Where is an acceptable compromise? And what does acceptable mean?

This chapter is organized in the following way: In Sect. 4.2 the origin of superluminous information transport on a macroscopic scale is described in some detail and some first steps to eliminate it are being suggested. Sect. 4.3 moves toward

Chapter 4

Causality Violations in Cascade Models of Nuclear Collisions

4.1 Introduction

Experiments like the upcoming nucleus-nucleus collisions at Brookhaven's Relativistic Heavy Ion Collider (RHIC) will without doubt open new possibilities to study the properties of nuclear matter at extreme pressures and temperatures. One major focus of the research is the question of whether or not there is a phase transition between hadronic and quark-gluon matter. The major open question that comes with this undertaking is: If there is a quark-gluon plasma, what will it look like in the detectors? What are its experimental signatures? The latter question can only be answered reliably if a theoretical model for the collision processes that goes beyond a phenomenological description is in place. One possible microscopic approach is to extend the semiclassical transport theory (as for example described in Chapter 3) to high-energy physics[Kal93, Sor89.1, Sor89.2, Gei92.1, Gei94.1, Gei92.2, Gei93].

Simulations of ultrarelativistic heavy ion collisions inevitably involve both soft and hard processes. Below some energy scale, soft or nonperturbative phenomena necessitate phenomenological description. Uncertainty is duly noted but for the foreseeable future a rigorous theory for soft physics is beyond understanding. However, through-microscopic physics and provides mathematical details for origins of unphysical effects. Whenever cross sections are finite and action-at-a-distance influences particle trajectories these problems inevitably arise. Sect. 4.4 deals with the resulting frame-of-reference dependence of the simulation. In Sect. 4.5 the used version of a parton cascade implementation is described. Then in Sect. 4.6 several different schemes which reduce or eliminate superluminous transport are compared. Methods include simply blocking collisions or truncations, scaling the cross sections downward while increasing the number of particles, and suppressing low-energy collisions. We also consider so-called wee-partons as a different way to define the initial conditions of the simulation, and point out how much this scheme affects causality violating mechanisms. Finally, Sect.4.7 concludes by briefly summarizing, and by discussing the outlook for future studies.

4.2 Macroscopic Causality Violations

Superluminous macroscopic information transport occurs mainly in the transverse (perpendicular to the beam) direction. The signal can travel over the diameter of the cross section σ in a single timestep and then continue this propagation from timestep to timestep. Transverse signal velocities can therefore reach $\sqrt{\frac{\sigma}{\pi}}/\Delta t_{\text{step}}$ over several timesteps, one is reminded of a chain of falling dominoes. The situation is depicted in Figure 4.1 for such a transport. As part of the model the two initially scattered particles cannot scatter again in this timestep. However, one timestep later that scattered particle from the right scatters with another particle coming from the right. As a result information from the first scattering, such as particle momenta and type, has travelled to the second one. This is a general problem of all transport codes, and is worsened because the gluon-gluon cross section becomes rather large for low energies (instead of vanishing). Along with the large gluonic cross sections are

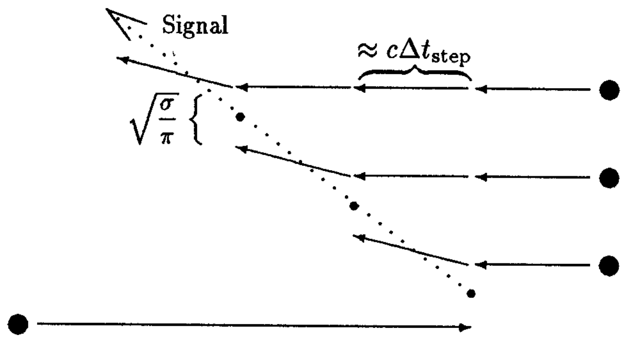

Signal
$\approx c\Delta t_{\text{step}}$
$\sqrt{\frac{\sigma}{\pi}}$

Figure 4.1: One particle coming from the left scatters with another particle coming from the right. The information from this event is rapidly passed on through a chain of subsequent scattering events.

relatively large gluon densities which result in very high probabilities for scattering in subsequent timesteps. Therefore information transport could be supported over relatively large distances without damping.

In many existing cascade codes first steps for dealing with the problem of superluminous signals have been taken: Most of the codes, including the one presented, allow only one interaction per particle per timestep. This restriction is consequent and clearly justified since a timestep is the shortest scale in the model. This restriction prevents signals from avalanching over huge volumes within only one timestep.

A second restriction implemented in many codes is the "closest approach" criterion. For a scattering process it demands in addition to "spatial distance within total cross section" also "the two particles have reached their point of closest approach assuming their current trajectories". Usually one would look at $|x^\mu x_\mu|$ with x_μ being the four-vector distance between the two particles involved, and demand that this quantity is minimal. As the two particles' positions are taken in the same timestep this quantity reduces to the spatial distance squared. Even though this looks like a Lorentz-invariant criterion, it is not, since the conceptual necessity of taking both particles' position at the beginning of a timestep (i.e. with vanishing time-separation in the lab-frame) is a non-Lorentz invariant restriction, see section 4.4. Nevertheless, the "closest approach" restriction prevents causality violating signal transport in the longitudinal (parallel to the beam) direction, but unfortunately has no effect on the transverse direction.

To demonstrate those causality violating shock-waves, the parton cascade simulation of a 100 GeV/nucleon (p,Au)-collision was run with different constant cross sections and S-wave scattering, the distance of of scattering events from the beam axis versus the simulation time is shown in Figure 4.2. The solid line indicates the distance that could be reached by a signal that travels with the speed of light. One can clearly see the outwards travelling shock-wave and how it gets damped out with time. For the smaller cross sections the causality violation is rather small and only present in the initial stages of the interaction. One should note that for realistic energy-dependent cross sections the initial scattering events take place with a lower cross section than the later ones because the c.m.-energy is much higher.

In the lower right panel the maximum distance $d_{\max}$ from the beam axis at which a scattering event occured during the simulation is plotted versus the cross section σ. The horizontal line indicates a distance that during the simulation time could only be reached with the speed of light.

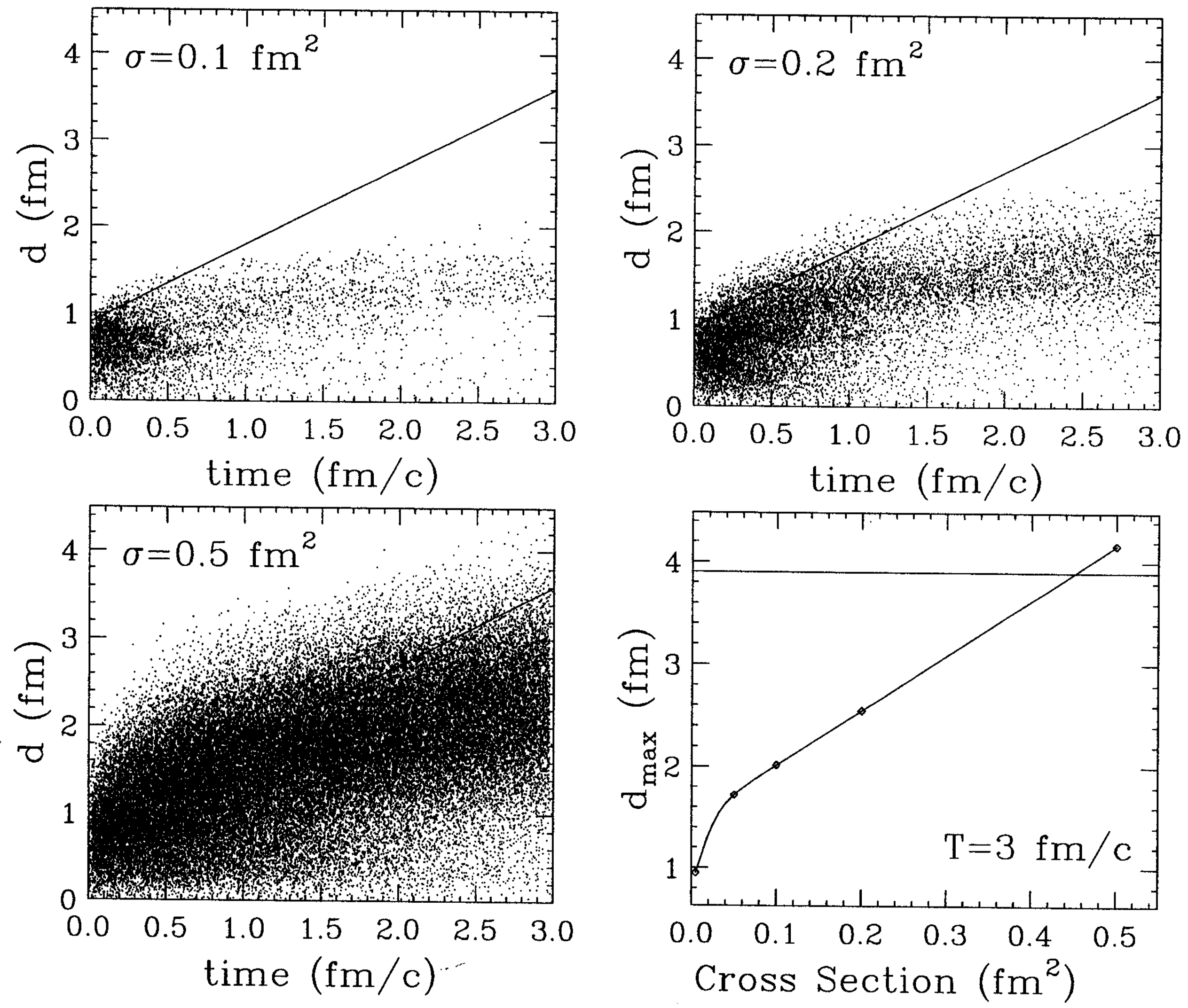


Figure 4.2: Distance d of scattering events versus simulation time at constant cross sections of σ =0.1, 0.2 and 0.5 fm^2.

While during the initial stages of the collisions the outmost scattering events occur at distances larger than those allowed by causality arguments, the information transport soon appears to be damped. It turns out that this phenomenon is not – as one would expect – mainly due to a dropping of the collision rate, but rather to a form of random-walk: only a fraction of the individual signal propagations lead outwards, others have the opposite effect, which results in an effective damping. In fact it turns out that the expectation value of the distance of scattering events from the beam axis is roughly proportional to the square-root of the simulation time, which is a characteristic feature of random-walk mechanisms, compare the top panels of Figure 4.2. This mechanism is a valid description until the cross sections are so large that the parton distribution basically appears solid, in that limit the information travels outwards proportional to the simulation time, see the panel for $\sigma = 0.5$ fm^2. By the nature of random-walks, the information expands fastest in the initial phase of the collision. With realistic energy-dependent cross sections however, those initial scattering events happen at lower cross sections than the ones in the later stages due to the higher c.m.-energy, which partly suppresses this initial outburst. Overall for σ=0.2 fm^2 the causality violations are still rather moderate, for 0.5 fm^2 the effect becomes dominant.

As already pointed out, Figure 4.2 also shows the maximum distance of scattering events from the beam axis as they occurred for different cross sections. As it turns out, this value depends linearly on the cross section for an extended range, beyond $\sigma \approx 0.4$ fm^2 the outmost scattering events occur at a distance that can only be reached with superluminous signal velocities.

4.3 Microscopic Causality Violations

In addition to macroscopic causality violations such as superluminous shock-waves, both the non-retarded interactions and the model inherent propagation of the whole particle configuration from one timestep to the next lead to causality violations on the level of elementary scattering processes.

A Lorentz-invariant simulation of parton scattering would require a truly 4-dimensional configuration space, in which an interaction can be established between any space-time point of one parton's trajectory and any other space-time point on another parton's trajectory. In the framework of classical fields, the scattering of two particles would be represented by their continuous change of trajectory in the retarded field of the respective other particle.

The actual realization of scattering events in the simulation only agrees in a certain limit with this model, namely if the two partons are coming from infinity with infinite rapidity and opposite directions: with increasing rapidity, in the lab-frame the fields are distorted to pancakes with an orientation perpendicular to the trajectory, reducing the dominant part of the interaction between the particles to a shorter and shorter region along their trajectories. In the limit of infinite rapidity, the interaction between two partons travelling with a non-vanishing impact parameter in opposite directions would be reduced to a momentary interaction at the point of their closest approach, making them change their trajectories at equal-times. This interaction would take place at a spacelike distance equal to the impact parameter – why does this not violate causality? The reason is, that the parton's field-pancake – even though it is travelling along with its present position – is build up from contributions out of the parton's past, the respective other parton does not scatter from the field generated at the first parton's present position, but from a field component that was generated along the

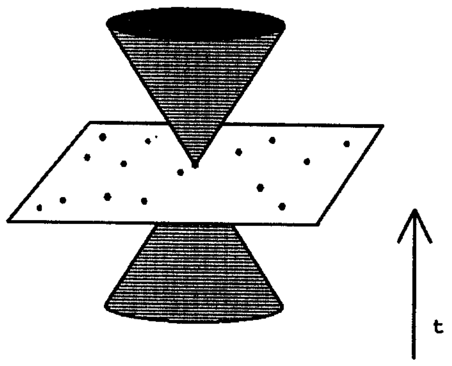


Figure 4.3: Lightcone and 3-dimensional hyperplane (2-dimensional illustration)

first parton's trajectory at a lightlike distance.

This limit agrees both with the claim of Refs. [Sor89.1, Sor89.2] that the interaction distance should be spacelike, because otherwise an interaction would influence the absolute past of one of the partons, and with the actual realization of scattering processes in the simulation: in the framework of a 3+1 dimensional transport simulation the distance of the testparticles is by the model itself determined to be spacelike: The information available at the beginning of a timestep is no more than the points of the trajectories on a certain x, y, z-hyperplane, as well as the corresponding momentum 4-vectors of the particles, also only at that time-coordinate. The lightcone of any one particle extends both before and after that hyperplane (see Fig. 4.3), but has no extension into that hyperplane itself, causing all other particles present in the simulation to have a spacelike distance from it.

But this limit soon loses its applicability: Staying in the model of interactions between the partons due to classical fields, as already pointed out, the field travelling with the partons is in fact build up from contributions out of their history, which for partons coming from infinity results in Lorentz-contracted field-pancakes.

But in a cascade simulation it is highly unlikely that seen from a parton A, parton B already was on the same trajectory a lightlike distance ago. In fact, as the partons are nearly travelling with the speed of light themselves, spacetime points with lightlike distances along their trajectories can happen to be very long ago, which in connection with the high interaction rates makes the idealized picture of the field pancakes inapplicable. Changes in trajectories can make more than one point along the trajectory of parton B have a lightlike distance to parton A, the acceleration connected to the changes in trajectory of parton B generates additional fields. Even worse, through the mechanism of parton generation and absorption, parton B might not even have existed at lightlike distances from parton A, still within the framework of the transport simulation, scattering would be possible – and causality violating.

Also, as in the idealized picture the c.m.-frame is moving parallel to the partons' trajectories and perpendicular to their distance at the point of closest approach, the equal-time character is true in both the lab- and the c.m.-frame. As soon as the two partons are not travelling into opposite directions, their spacial distance also leads to a time-separation, making the scattering time different in the lab- and c.m.-frame.

4.4 Frame-Dependencies and Time-Ordering

An obvious consequence of the model's causality violations is its frame-of-reference dependence: The simulation will for example certainly lead to different results when run in the rest-frame of the target. This was already shown for the example of a $^{12}C + ^{12}C$ internuclear cascade calculation in Ref. [Kod84], the authors demonstrated that both the total number of collisions and the individual time-ordering of given collisions were frame-dependent. As obvious as that seems from the simple problems pointed out in sections 4.2 and 4.3, for a parton cascade it is not yet the whole story:

The simulation cannot even be run in the rest-frame of any one nucleus, because there is no possibility to create appropriate initial conditions. The parton distribution in nucleonic matter is frame-dependent, the higher the energy of a nucleon the higher the number of virtual partons. To put it in other words, usually components of the nucleonic wavefunction are considered a parton if they carry a momentum fraction beyond a threshold given by a minimum x-value, the latter one being a parameter of the model. However, by boosting the nucleonic wavefunction into an accelerated frame of reference, a procedure described by the model-parameter Q^2 of the parton distribution function $f(x, Q^2)$, more and more components of it cross that threshold x-value, leading to higher and higher parton numbers.

Therefore, when asking questions about how the collision of two partons looks in their rest-frame as compared to the lab-frame, another valid question is: Do those two partons even exist in the other frame? Or are in that frame other partons around that the collision partners would be much more likely to scatter with instead?

However, as this question cannot be addressed within the framework of a transport simulation, it is worthwhile to study the effects of a transformation of two given partons into their c.m.-frame.

4.4.1 The Transformation Equations

The transformation of a four vector $(a_0, \boldsymbol{a})$ with a boosting vector $\boldsymbol{\beta}$ and $\gamma := \frac{1}{\sqrt{1-\beta^2}}$ is given by the following set of equations:

$$a_{0\mathrm{cm}} = \gamma(a_{0\mathrm{lab}} - \boldsymbol{\beta}\boldsymbol{a}_{\mathrm{lab}}) \tag{4.1}$$

$$\boldsymbol{a}_{\mathrm{cm}} = \boldsymbol{a}_{\mathrm{lab}} + \frac{\gamma-1}{\beta^2}(\boldsymbol{\beta}\boldsymbol{a}_{\mathrm{lab}})\boldsymbol{\beta} - \gamma\boldsymbol{\beta}a_{0\mathrm{lab}} \tag{4.2}$$

$$a_{0\mathrm{lab}} = \gamma(a_{0\mathrm{cm}} + \boldsymbol{\beta}\boldsymbol{a}_{\mathrm{cm}}) \tag{4.3}$$

$$\boldsymbol{a}_{\mathrm{lab}} = \boldsymbol{a}_{\mathrm{cm}} + \frac{\gamma-1}{\beta^2}(\boldsymbol{\beta}\boldsymbol{a}_{\mathrm{cm}})\boldsymbol{\beta} + \gamma\boldsymbol{\beta}a_{0\mathrm{cm}} \tag{4.4}$$

4.4.2 Finding the Boost Parameters

The c.m.-frame is defined by

$$\boldsymbol{p}_{\mathrm{cm},1} = -\boldsymbol{p}_{\mathrm{cm},2} , \tag{4.5}$$

$\boldsymbol{p}_{\mathrm{cm},1}$ and $\boldsymbol{p}_{\mathrm{cm},2}$ being the momenta of the partons in the c.m.-frame. This leads to

$$\boldsymbol{\beta} := \frac{\boldsymbol{p}_{\mathrm{lab},1} + \boldsymbol{p}_{\mathrm{lab},2}}{E_{\mathrm{lab},1} + E_{\mathrm{lab},2}} . \tag{4.6}$$

4.4.3 Transformation of Momenta and Energy

Using equation (4.1) with $\boldsymbol{\beta}$ from equation (4.6) gives the energies of the partons in the c.m.-frame:

$$E_{\mathrm{cm},1} = \gamma(E_{\mathrm{lab},1} - \boldsymbol{\beta}\boldsymbol{p}_{\mathrm{lab},1}) \tag{4.7}$$

$$E_{\mathrm{cm},2} = \gamma(E_{\mathrm{lab},2} - \boldsymbol{\beta}\boldsymbol{p}_{\mathrm{lab},2}) . \tag{4.8}$$

The relative momentum can be calculated by means of equation (4.2) with both $\boldsymbol{p}_{\mathrm{lab},1}$ and $\boldsymbol{p}_{\mathrm{lab},2}$, however, parton 1 is chosen:

$$\boldsymbol{p}_{\mathrm{cm}} := \boldsymbol{p}_{\mathrm{lab},1} + \frac{\gamma - 1}{\beta^2}(\boldsymbol{\beta}\boldsymbol{p}_{\mathrm{lab},1})\boldsymbol{\beta} - \gamma\boldsymbol{\beta}E_{\mathrm{lab},1} \tag{4.9}$$

The total energy $\sqrt{s}$ in the c.m.-frame is given by means of equation (4.3) and the four vector of the total momentum in the c.m.-frame $(E_{\mathrm{cm},1} + E_{\mathrm{cm},2}, \boldsymbol{p}_{\mathrm{cm},1} + \boldsymbol{p}_{\mathrm{cm},2}) =: (E_{\mathrm{cm}}, 0)$:

$$\sqrt{s} := E_{\mathrm{cm}} = \frac{E_{\mathrm{lab},1} + E_{\mathrm{lab},2}}{\gamma} \tag{4.10}$$

4.4.4 Transformation of the Relative Positions

The spatial distance between the two partons is defined as the vector from parton 2 to parton 1. The time separation is to be assumed between zero and the timestep length:

$$\Delta \boldsymbol{r}_{\text{lab}} := \boldsymbol{r}_{\text{lab},1} - \boldsymbol{r}_{\text{lab},2} \tag{4.11}$$

$$0 \leq \Delta t_{\text{lab}} \leq \Delta t_{\text{step}} \tag{4.12}$$

If however one assumes the particles' positions to be taken at a certain moment inside of the timestep interval, for example at the beginning, the time separation Δt_{lab} is zero.

The distance vector in the c.m.-frame is then given by

$$\Delta \boldsymbol{r}_{\text{cm}} = \Delta \boldsymbol{r}_{\text{lab}} + \frac{\gamma - 1}{\beta^2}(\boldsymbol{\beta} \Delta \boldsymbol{r}_{\text{lab}})\boldsymbol{\beta} \, . \tag{4.13}$$

In addition to that, the partons will now have a time separation in the c.m.-frame:

$$\Delta t_{\text{cm}} = -\gamma \boldsymbol{\beta} \Delta \boldsymbol{r}_{\text{lab}} \tag{4.14}$$

As soon as this time-separation is larger than the transformed timestep length, Lorentz-invariance is clearly violated: The partons' positions are not taken within the same timestep inside the c.m.-frame anymore.

4.4.5 Ways to Calculate the Positions of the Partons in the Next Timestep

Propagation Within the c.m.-Frame

A first idea is to propagate the partons in their c.m.-frame. The distance vector then changes to

$$\Delta \boldsymbol{r}'_{\text{cm}} = \Delta \boldsymbol{r}_{\text{cm}} + \left(\frac{\boldsymbol{p}_{\text{cm}}}{E_{\text{cm},1}} + \frac{\boldsymbol{p}_{\text{cm}}}{E_{\text{cm},2}} \right) \Delta t_{\text{step,cm}} \, . \tag{4.15}$$

In order to have the time $\Delta t_{\text{step,cm}}$ correspond to a four vector with the time component Δt_{step} in the lab frame, one has to choose

$$\Delta t_{\text{step,cm}} = \frac{\Delta t_{\text{step}}}{\gamma} \, , \tag{4.16}$$

and therefore

$$\Delta \boldsymbol{r}'_{\mathrm{cm}} = \Delta \boldsymbol{r}_{\mathrm{cm}} + \left(\frac{\boldsymbol{p}_{\mathrm{cm}}}{\gamma E_{\mathrm{cm},1}} + \frac{\boldsymbol{p}_{\mathrm{cm}}}{\gamma E_{\mathrm{cm},2}} \right) \Delta t_{\mathrm{step}} \ . \tag{4.17}$$

This strategy however completely neglects the spatial components of the timestep that occur during the reverse transformation.

Propagation Within the Lab-Frame

A more refined strategy is the propagation of the partons in the lab-frame and subsequent transformation of the new positions to the c.m.-frame. The propagation vectors for the partons take the form

$$\left(\Delta t_{\mathrm{step}},\ \frac{\boldsymbol{p}_{\mathrm{lab,i}}}{E_{\mathrm{lab,i}}} \Delta t_{\mathrm{step}} \right) \ .$$

With $\boldsymbol{\Lambda}_{a_0}(\boldsymbol{a})$ being the spatial Lorentz-transformation into the c.m.-frame for a vector $(a_0, \boldsymbol{a})$, one gets

$$\begin{aligned} \Delta \boldsymbol{r}'_{\mathrm{cm}} &= \boldsymbol{\Lambda}_{0+\Delta t_{\mathrm{step}}-0-\Delta t_{\mathrm{step}}} \left(\boldsymbol{r}_{\mathrm{lab},1} + \frac{\boldsymbol{p}_{\mathrm{lab},1}}{E_{\mathrm{lab},1}} \Delta t_{\mathrm{step}} - \boldsymbol{r}_{\mathrm{lab},2} - \frac{\boldsymbol{p}_{\mathrm{lab},2}}{E_{\mathrm{lab},2}} \Delta t_{\mathrm{step}} \right) \quad (4.18) \\ &= \Delta \boldsymbol{r}_{\mathrm{cm}} + \boldsymbol{\Lambda}_0 \left(\frac{\boldsymbol{p}_{\mathrm{lab},1}}{E_{\mathrm{lab},1}} \Delta t_{\mathrm{step}} - \frac{\boldsymbol{p}_{\mathrm{lab},2}}{E_{\mathrm{lab},2}} \Delta t_{\mathrm{step}} \right) \\ &= \Delta \boldsymbol{r}_{\mathrm{cm}} + \boldsymbol{\Lambda}_0 \left(\frac{\boldsymbol{p}_{\mathrm{lab},1}}{E_{\mathrm{lab},1}} \Delta t_{\mathrm{step}} \right) - \boldsymbol{\Lambda}_0 \left(\frac{\boldsymbol{p}_{\mathrm{lab},2}}{E_{\mathrm{lab},2}} \Delta t_{\mathrm{step}} \right) \ . \end{aligned}$$

Because of

$$\boldsymbol{\Lambda}_0(\boldsymbol{p}_{\mathrm{lab,i}}) = \boldsymbol{\Lambda}_{E_{\mathrm{lab,i}}}(\boldsymbol{p}_{\mathrm{lab,i}}) + \gamma \boldsymbol{\beta} E_{\mathrm{lab,i}} = \boldsymbol{p}_{\mathrm{cm,i}} + \gamma \boldsymbol{\beta} E_{\mathrm{lab,i}} \tag{4.19}$$

the result is

$$\Delta \boldsymbol{r}'_{\mathrm{cm}} = \Delta \boldsymbol{r}_{\mathrm{cm}} + \left(\frac{\boldsymbol{p}_{\mathrm{cm}}}{\gamma (E_{\mathrm{cm},1} + \boldsymbol{\beta} \boldsymbol{p}_{\mathrm{cm}})} + \frac{\boldsymbol{p}_{\mathrm{cm}}}{\gamma (E_{\mathrm{cm},2} - \boldsymbol{\beta} \boldsymbol{p}_{\mathrm{cm}})} \right) \Delta t_{\mathrm{step}} \ . \tag{4.20}$$

The following Minkowski plots again show the above in one dimension. The dashed lines denote the world lines of the two partons, in one dimension their closest approach can only be an intersection. The two solid lines denote the coordinate system in the partons' c.m.-frame.

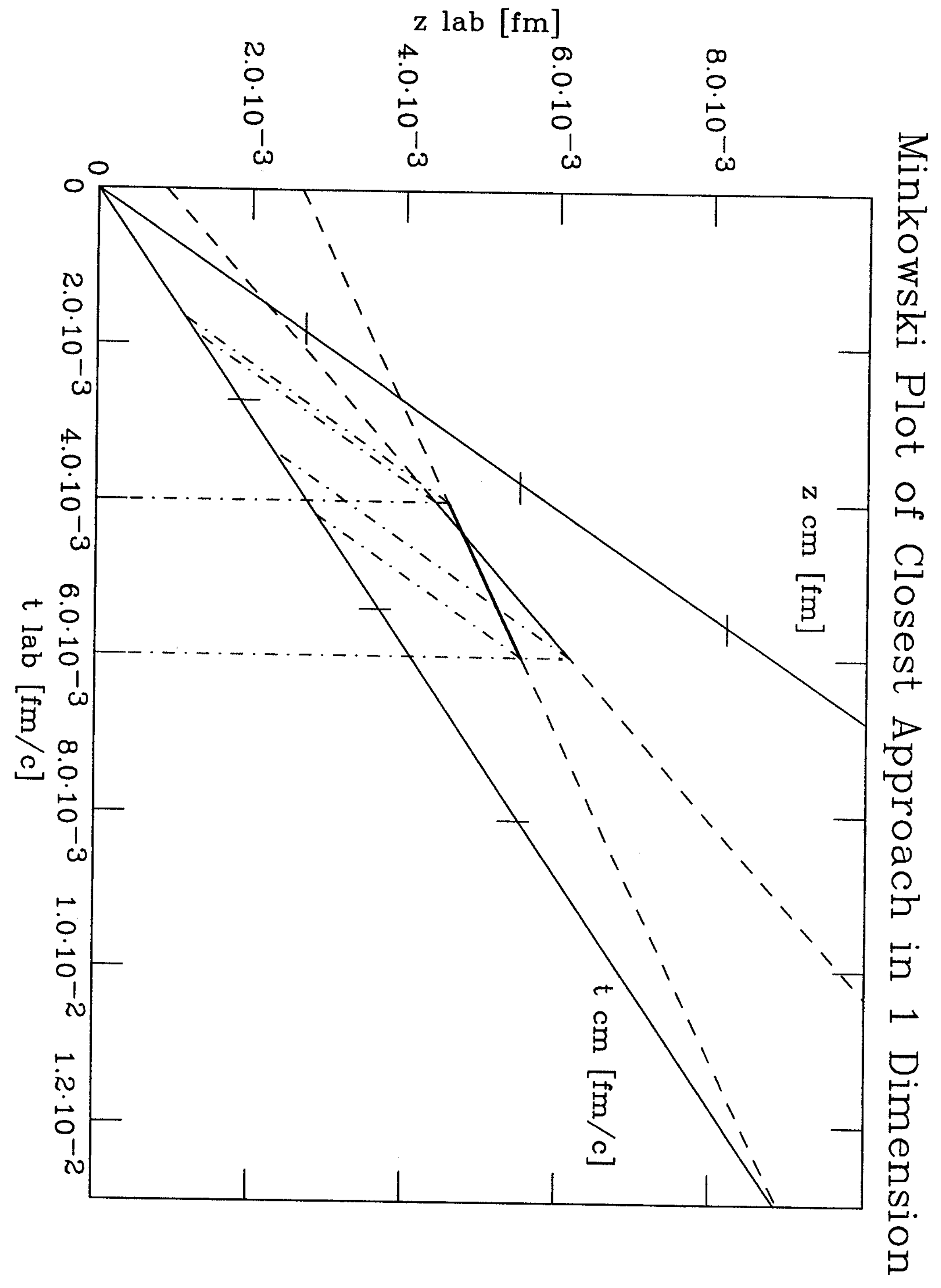


Figure 4.4: Minkowski Plot. Highlighted is the timestep in which the particles are at the point of closest approach. Note the time-order of the four events “Particle 1(2) at the beginning(end) of the timestep” in the c.m.-frame. These points in time are given by the intersection of the c.m.-t axis with the dot-dashed lines that are parallel to the c.m.-z axis.

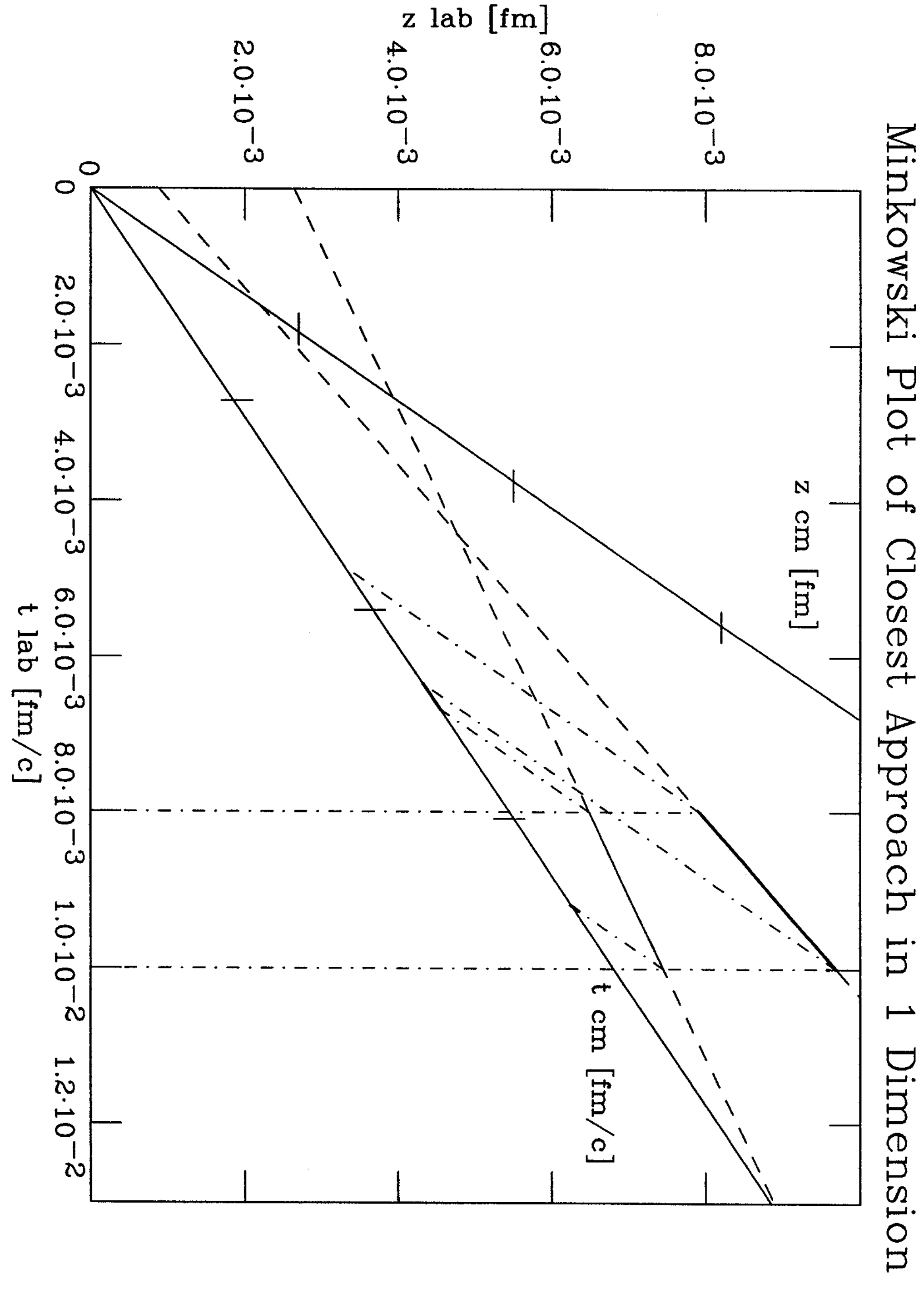


Figure 4.5: Minkowski Plot. Highlighted is the second timestep after the particles had reached their point of closest approach. Note the the time-order of the four events "Particle 1(2) at the beginning(end) of the timestep" in the c.m.-frame has changed.

The time step length in these examples is $2 \cdot 10^{-3}$ fm. The dot-dashed lines run through simultaneous time-space points in the respective frames. In figure 4.4 the time step with the point of closest approach is selected, one can clearly see the time separation of the two partons both at the beginning and at the end of the time step – this separation is rather small compared to the transformed time step length. However, two time steps (see Fig. 4.5) later those two time separations overlap, in the c.m.-frame the faster parton has reached it's position at the end of the lab frame's time step before the slower parton has reached it's position at the beginning of the lab frame's time step. The smaller the lab frame's time step the more likely this happens, which is counterintuitive to the usual fact that a simulation gets better with smaller timestep lengthes. In one dimension this reversed time-ordering is impossible in the time step of closest approach. In two or three dimensions this configuration can occur as soon as $\frac{|\Delta \boldsymbol{r}^{\text{c.a.}}_{\text{cm}}|}{\Delta t^{\text{c.a.}}_{\text{cm}}} \geq c$ at the point of the closest approach distance $|\Delta \boldsymbol{r}^{\text{c.a.}}_{\text{cm}}|$ – in other words, in two or three dimensions the reversed time-ordering is a problem even at the critical timestep of closest approach.

4.4.6 Consequences

As it had turned out, by boosting the partons with $\boldsymbol{\beta}$ into their c.m.-system, there they have a time-separation of

$$\Delta t_{\text{cm}} = -\gamma \boldsymbol{\beta} \cdot \Delta \boldsymbol{r}_{\text{lab}} \, . \tag{4.21}$$

The partons' positions, being taken at equal time in the lab-frame, will, unless $\boldsymbol{\beta}$ and $\Delta \boldsymbol{r}_{\text{lab}}$ are perpendicular, not be at equal time in their c.m.-frame and vice versa. The construction of a "Lorentz-invariant CMS distance" as in Refs. [Sor89.1, Sor89.2] seems doubtful. This time-separation can very well be larger than the transformed timestep length, leading to a situation where the partons' positions are not taken

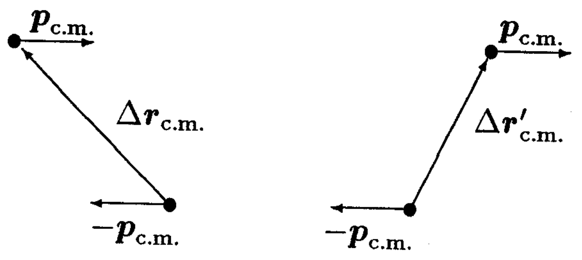


Figure 4.6: Two partons passing their point of closest approach in the c.m.-frame.

within the same timestep within the c.m.-frame anymore. As a result, the closest approach criterion can only provide a means to get an averaged point in time for an actual scattering event.

For this criterion, described in section 4.2, one can choose to define it either in the particles' c.m.-frame or in the lab-frame (a third approach will be described in subsection 4.6.6). Because of the non-vanishing time-separation of the partons in their c.m.-frame, the two possibilities are not equivalent. Both methods had been compared, but nevertheless no significant change in the collision rate was found, even though the individual collisions happening in both simulations are different. For further simulations the c.m.-frame was chosen, since intuitively this frame seems to be more significant for the individual scattering events. In this frame, the scalar products of the momentum and the distance vector both at the beginning and at the end of the timestep are to be calculated. If this scalar product changes sign, the position of closest approach is reached within this timestep. In Figure 4.6 $\Delta \boldsymbol{r}_{\mathrm{cm}}$ denotes the distance vector at the beginning of the timestep, $\Delta \boldsymbol{r}'_{\mathrm{cm}}$ at the end. The criterion is $(\boldsymbol{p}_{\mathrm{cm}} \cdot \Delta \boldsymbol{r}_{\mathrm{cm}})(\boldsymbol{p}_{\mathrm{cm}} \cdot \Delta \boldsymbol{r}'_{\mathrm{cm}}) \leq 0$.

The non-vanishing c.m.-time-separation unfortunately also makes the calculation of $\Delta \boldsymbol{r}'_{\mathrm{cm}}$ ambiguous.

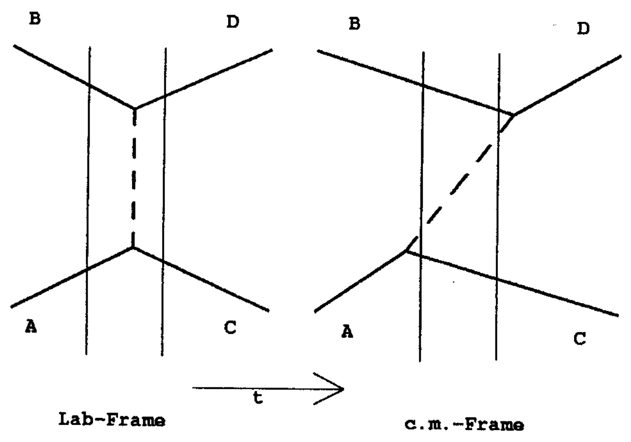


Figure 4.7: Possible discrepancies in the time-ordering of scattering processes that are usually prevented by the spacelike distance of the scattering partners.

In the model used, the c.m.-positions at the end of the timestep are calculated by propagating the partons within the lab-frame and boosting the result into the c.m.-frame. The result, expressed in c.m.-quantities, is

$$\begin{aligned} \Delta \boldsymbol{r}'_{\text{cm}} &= \Delta \boldsymbol{r}_{\text{cm}} \\ + & \left(\frac{\boldsymbol{p}_{\text{cm}}}{\gamma(E_{\text{cm},1} + \boldsymbol{\beta} \cdot \boldsymbol{p}_{\text{cm}})} + \frac{\boldsymbol{p}_{\text{cm}}}{\gamma(E_{\text{cm},2} - \boldsymbol{\beta} \cdot \boldsymbol{p}_{\text{cm}})} \right) \Delta t_{\text{step}} \, . \end{aligned} \tag{4.22}$$

Due to the spacelike distance of the partons however, the definition of "beginning" and "end of a timestep" becomes frame-dependent.

Consider a process where the partons A and B scatter and produce a parton pair C and D, see Figure 4.7. In the lab-frame this happens within one timestep, at the beginning of the lab-frame's timestep (denoted by thin vertical lines in the left panel) partons A and B are present, at the end partons C and D. In the transformed timestep within the c.m.-frame however (denoted by thin vertical lines in the right panel), this set-up can easily be distorted to a situation where in the beginning of that timestep partons B and C are present, and at the end partons C and D. Seen from yet another frame, in the beginning A and D could exist.

One consequence is that in the simulation re-scattering processes have to be explicitly forbidden: In spite of the closest-approach criterion re-scattering would still be possible if two partons are scattered towards each other – as the distance between partons involved in a scattering event can very well be greater than $\Delta t_{\text{step}}c$, it is possible for those two scattered partons to again reach a point of closest approach in a later timestep. In the case of a retarded interaction, this would not be possible.

More relevant then the time-scale given by the timestep, which is a model-dependent parameter, is the time-scale given for example by the time it takes for the two nuclei to cross through each other, which is about 0.2 fm/c. This time-interval corresponds to an impact parameter of 0.2 fm beyond which the time-ordering of scattering events is frame-dependent. As the interactions in question happen between particles travelling into the same direction, those would also be low-energy interactions with cross sections very well in the range of such impact parameters. The effects of this frame-dependent time-ordering of the incoming and outgoing particles of a scattering event have to be subject of extensive examination.

An interesting approach is that of Refs. [Lan93, Dan95]: In this method, configuration space is divided into boxes. Within the boxes the scattering partners are randomly chosen according to a probability that is a function of the cross section – convergence of this method towards the solution of the Boltzmann equation for an infinitely small box size and timestep length, as well as an infinite number of test-particles, is shown in Ref. [Bad89]. The code of Rev. [Dan95] was successfully made more efficient by only considering a fraction of the possible pair combinations within each box, and compensating the otherwise resulting loss in collision rate by an enhanced probability for the collision of the chosen pairs. The authors of Ref. [Lan93] claim that this stochastic method of formulating the collision term is covariant since it is dealing with transition rates instead of geometrical interpretations, therefore no problems connected with the time-ordering of processes would occur. In fact, since in the model the time-order of processes is chosen randomly anyway, the model has no "right" time-order that could be distorted by relativistic effects. However, this new approach does not overcome the problem of superluminous shock-waves. Within a given cell of longitudinal size $\delta r_{\|}$ and perpendicular size $\delta r_{\perp}$ any two particles have a chance of colliding. Since the subsequent advection step may carry some of these into neighboring cells, the maximum transverse velocity for information transport will be $\delta r_{\perp}/\Delta t_{\mathrm{step}}$. Since in general $\delta r_{\perp}^2$ is several times larger than the cross section σ, the resulting maximum possible causality violations in transverse direction are even larger than in the method discussed above. In addition, the stochastic method also allows for superluminous information propagation in longitudinal direction, with velocity $\delta r_{\|}/\Delta t_{\mathrm{step}}$.

4.5 The Parton Model

Classical simulations, which have been successfully applied to heavy ion collisions at intermediate energies [Ber88], are now [Kal93, Sor89.1, Sor89.2, Gei92.1, Gei94.1, Gei92.2, Gei93] being extended to high energies. The main step in the extension of this microscopic model is using a parton based picture of the nuclei rather than a hadronic picture; consequently the interactions between the testparticles are to be described in the framework of QCD, leading to so-called parton cascades.

The code works in 3+1 dimensions using fully relativistic kinematics for the partons, where the quarks are consequently treated as massive particles. Both quarks and gluons can be off-shell. The initial conditions are determined by standard parton distribution functions $f(x, Q^2)$ [Bot93], where the value for Q^2 and the minimum x are parameters of the model. Technically stability of the incoming nuclei is guaranteed both by a coherent motion of all partons in longitudinal direction, and by the

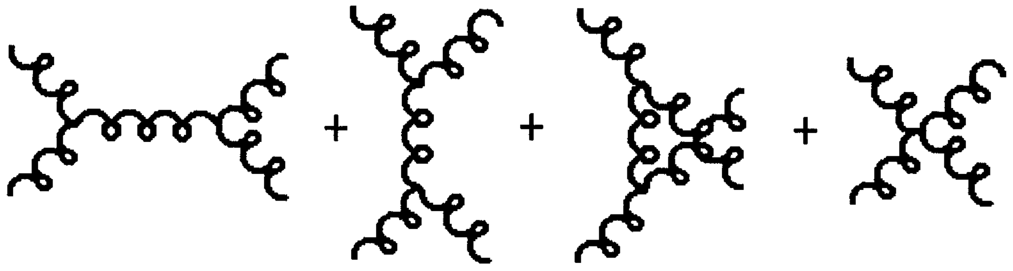


Figure 4.8: Typical contributions to the gluon-gluon cross section. The four point diagram on the far right leads to divergencies in the cross section that cannot be handled by phenomenological cut-off masses.

restriction that particles from the same nucleus cannot scatter with each other before at least one of them has scattered with a particle from the other nucleus.

In this preliminary version of the code only QCD processes with two partons in the incoming and two partons in the outgoing state are implemented [Mur93]. Phenomenological screening or cut-off masses have been added into the propagators to avoid divergent total cross sections. However, the gluon-gluon scattering cross section includes a four point diagram which does not contain a propagator, Fig. 4.8 shows typical contributions to the gluon-gluon cross section.

This means that the divergence in this cross section must be handled differently from the other cross sections. One usually regularizes the gluon-gluon cross section by setting it to a constant value below a certain cut-off energy, in this case a a cutoff of $s \approx 0.25$ GeV^2 has been chosen corresponding to $\sigma \leq 0.45$ fm^2. This cutoff seems appropriate because it agrees both with a reasonable c.m.-energy below which Perturbative QCD loses validity, and with the limiting cross section for superluminous shock-waves found in section 4.2. Other methods of cutting off these cross sections while maintaining the correct physics are being investigated. Also, no medium mod-

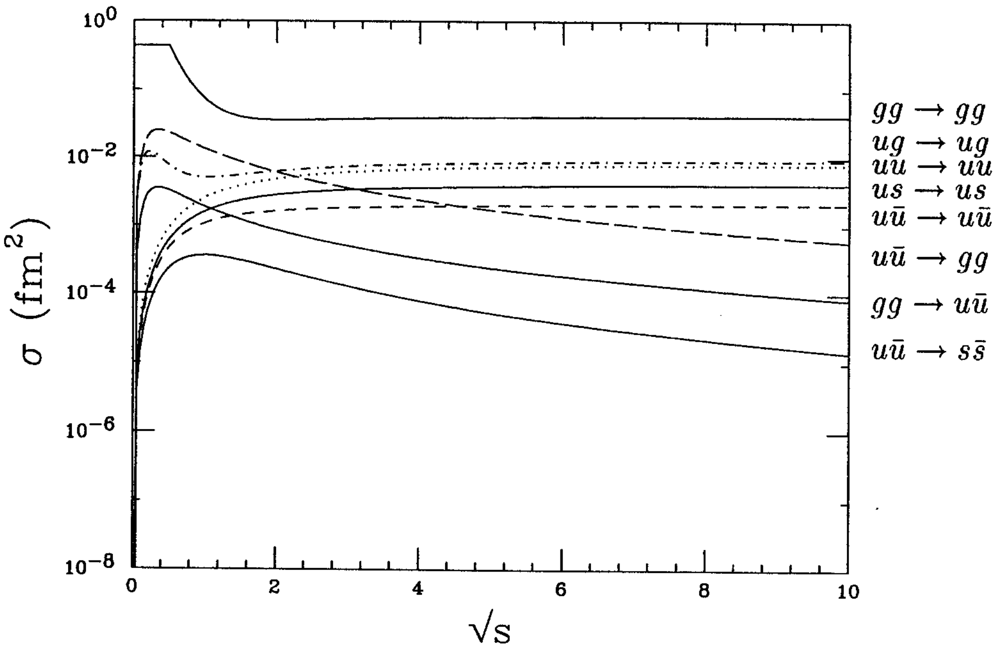


Figure 4.9: Example of partial cross sections being used versus the c.m.-energy.

ifications to the elementary cross sections are taken into account. Figure 4.9 shows the cross sections being used. The analytic expressions for cross sections have been checked in the massless quark limit against results from the literature [Com77], some of them also in the massive case [Eic84].

4.6 Comparison of Methods for Dealing with Superluminous Signals

This section contains a comparison of different methods for dealing with the problem of superluminous signal transport and is the outcome of simulating central collisions of a 100 GeV proton on a 100 GeV per nucleon gold nucleus.

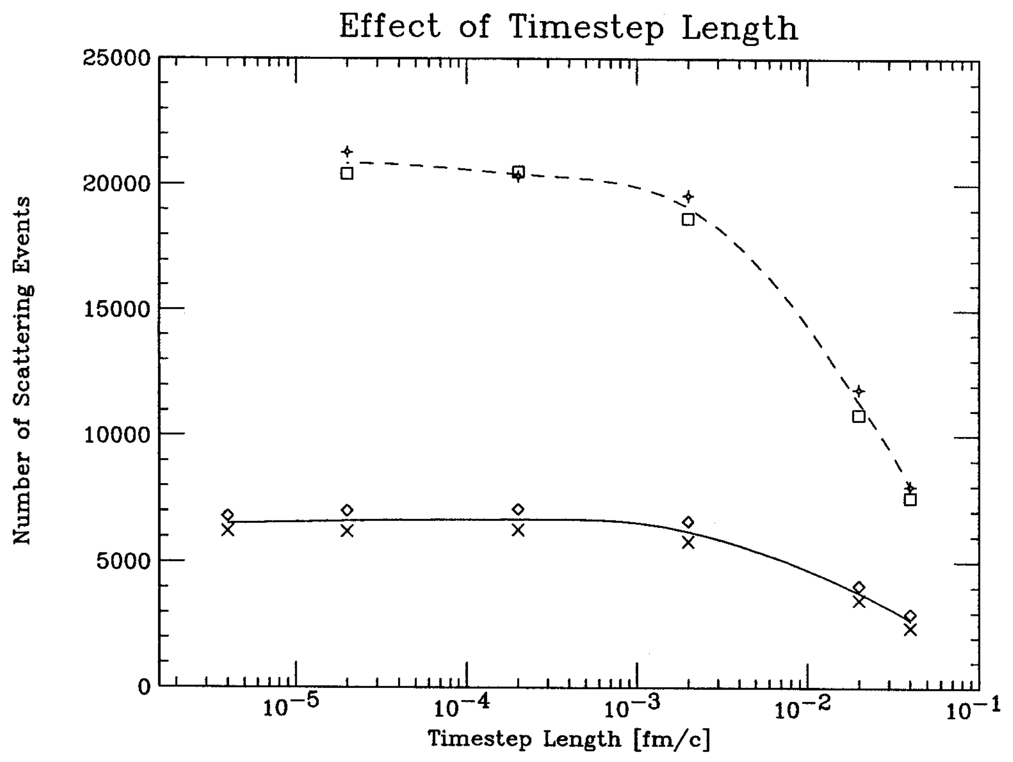


Figure 4.10: Total number of scattering events versus timestep length used for the simulation. The dashed line interpolates the results from simulations of (La,La) collisions, the dotted line from (Cu,Cu) collisions. The symbols denote the actual data points generated in the respective simulations.

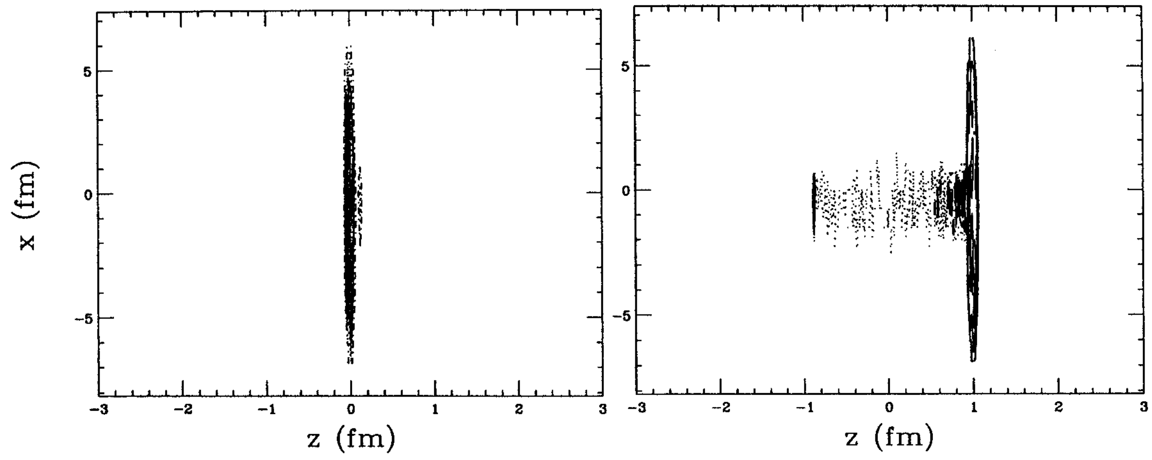


Figure 4.11: In the left panel the projection of the initial parton configuration on the x, z-plane is depicted. The highly Lorentz-contracted proton is moving to the left into the equally contracted gold nucleus which is moving to the right. The right panel shows the configuration 1 fm/c later.

This set-up seems suitable since it allows observation of the information transport from the incoming (comparably small) particle in nuclear matter. Table 4.1 summarizes the parameters used for the simulations. The timestep length of 0.0002 fm/c was determined by running the simulation with different timestep lengths and observing the total number of collisions. Figure 4.10 shows the outcome of those simulations, 0.0002 fm/c can be identified as the longest timestep length possible without leaving the plateau of nearly constant total collision numbers. Choosing the longest possible timestep length for which the simulation still converges minimizes the effect of causality violations due to "instantaneous" reactions over a finite distance.

With the values chosen for Q^2 and the minimum x-value approximately 9000 partons per testrun are generated. For the following calculations, the longitudinal extent of the nuclei in the model is determined by the Lorentz-contraction only, Figure 4.11 shows the initial configuration in the lab-frame and 1 fm/c later.

The effects of "wee-partons" are examined in the last subsection. The cross sec-

Table 4.1: Parameters used for the comparisons

Parameter	Chosen Value
Total time of simulation	3 fm/c
Timestep length	0.0002 fm/c
Number of timesteps	15000
Proton energy	100 GeV
Energy per nucleon Au	100 GeV
Impact parameter	0 fm
Minimum x-value for $f(x, Q^2)$	0.005
Q for $f(x, Q^2)$	25 GeV
Bag-radius for nucleons	0.9 fm
α_s	0.2
Cut-off mass for gluon propagators	1.0 GeV
Cut-off mass for quark propagators	0.2 GeV
Number of parallel test runs	5

tions were energy-dependent according to section 4.5 and Figure 4.9.

The simulation was first run for S-wave scattering in the expectation that this would be the worst case. Compared to the realistic forward peaked angular distributions, S-wave scattering favors transverse signal propagation. However, it turned out, that the angular distribution hardly makes any difference as far as causality violating effects are concerned, the only noticeable change was in the rapidity distributions, where it turned out that the gap in rapidity between unscattered and scattered partons was wider for S-wave scattering: the partons lose more units of rapidity in the initial collisions. The small effect of the angular distributions is understandable from the fact that even isotropic distributions in the c.m.-frame are strongly forward peaked in the lab-frame.

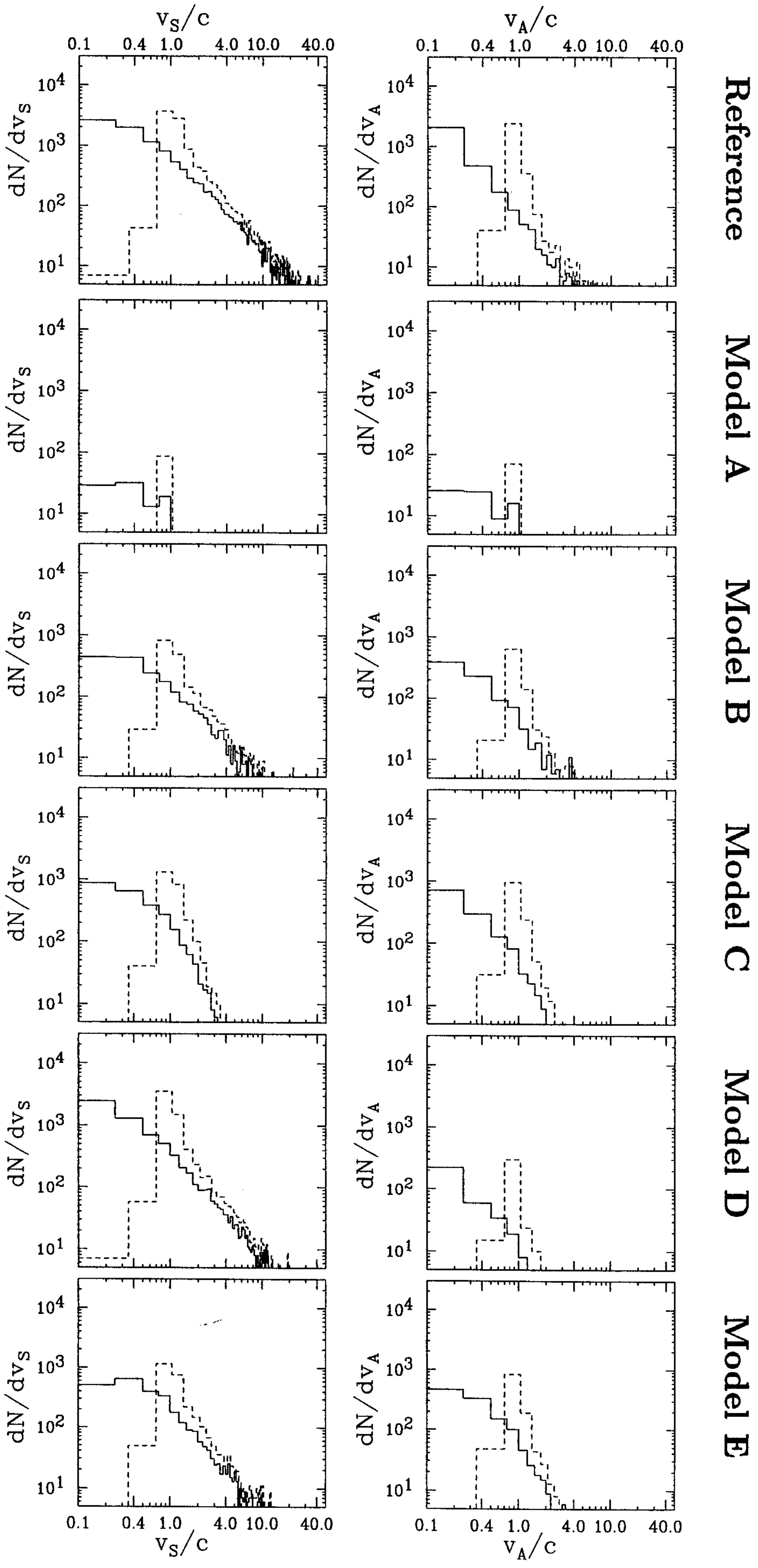
The top row of Figures 4.12,4.13 summarizes the results for a simulation with a realistic angular distribution. The top left histogram of Figure 4.12 shows the signal velocity distribution regarding directly subsequent collisions, see Figure 4.1. In other words, with $r^{\mu}_{n,i}$ being the position of the ith collision of the nth particle, the distribution of

$$v_S(n,i) := |\boldsymbol{r}_{n,i} - \boldsymbol{r}_{n,i-1}|/(t_{n,i} - t_{n,i-1}) \tag{4.23}$$

for all n and $i > 1$ is shown. The solid histogram shows the radial component of those velocities, the dashed histogram includes the longitudinal component – the strong peak at c results from processes for which the signal transport was dominated by the partons' motion with the respective nuclei. On the order of 50 isolated scattering events with velocities from $50c$ up to $1360c$ have been suppressed in the plot.

Figure 4.12: Results of simulations with different mechanisms to suppress superluminous signal transport (Part 1).

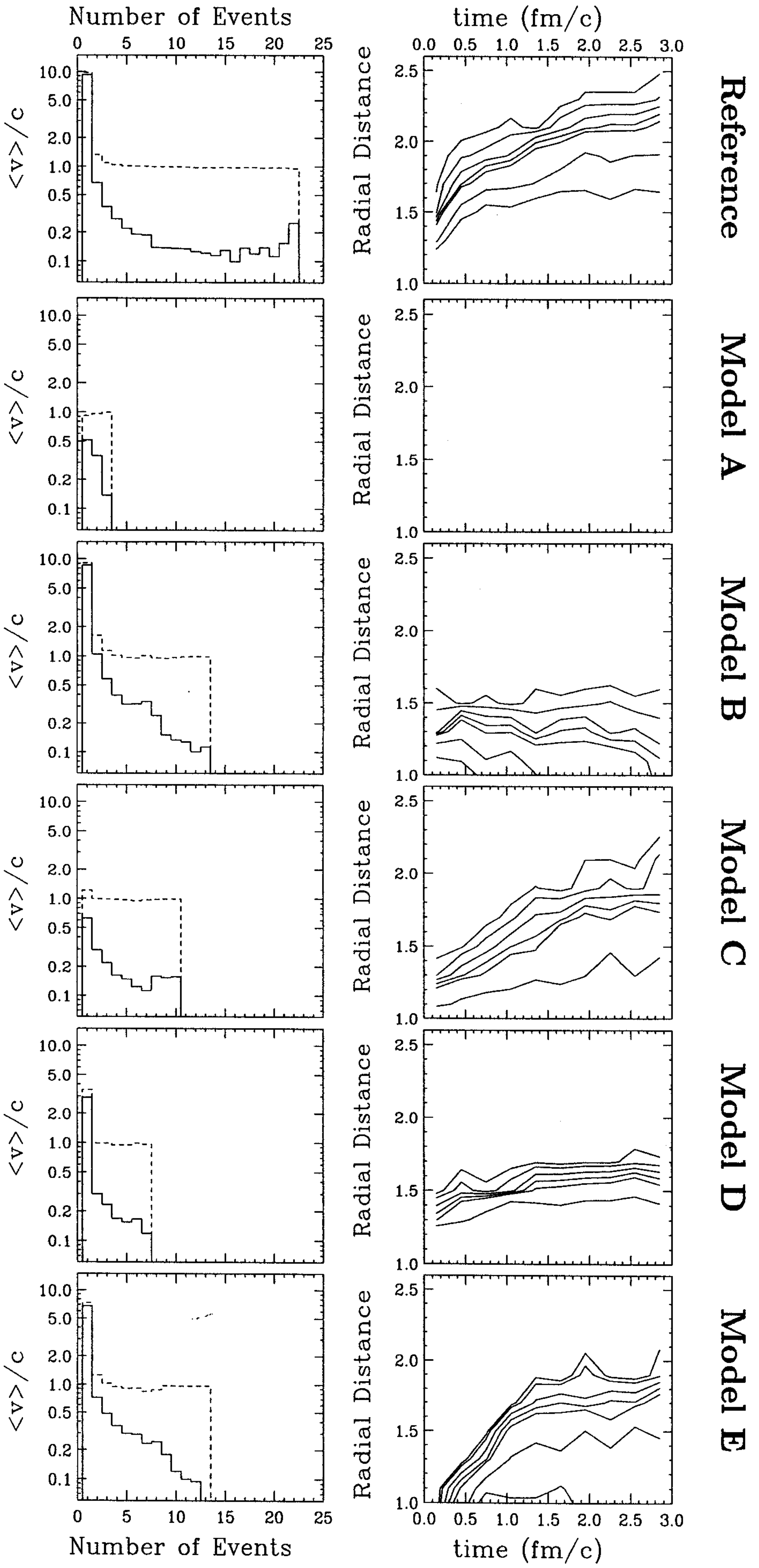
Figure 4.12



v_S/c
v_A/c
0.1
0.4
1.0
4.0
10.0
40.0
dN/dv_S
dN/dv_A
10^1
10^2
10^3
10^4
Reference
Model A
Model B
Model C
Model D
Model E

Figure 4.13: Results of simulations with different mechanisms to suppress superluminous signal transport (Part 2).

Figure 4.13


Number of Events
time (fm/c)
<v>/c
Radial Distance
Reference
Model A
Model B
Model C
Model D
Model E

The right panel shows a more general impression of the signal velocities. Instead of just taking into account the signal velocity occurring within one single scattering event, in this histogram the signal velocity from the very first to the very last scattering event that a parton was involved in is calculated. With the notation from above and $N(n)$ being the number of collisions that the parton n was involved in, the distribution of

$$v_A(n) := |\boldsymbol{r}_{n,N(n)} - \boldsymbol{r}_{n,1}|/(t_{n,N(n)} - t_{n,1}) \tag{4.24}$$

for all n is shown. Again, the solid histogram shows the radial component of the signal velocities, the dashed histogram also takes into account the longitudinal components.

Because of damping effects the average velocities are much smaller than the velocities from one event to the next, the left panel of Figure 4.13 shows how subsequent scattering events are leading to this overall damping of peak velocities occurring in single scattering events: again starting from the very first scattering event of a parton, the signal velocity to the ith following scattering event of the partons

$$v(n,i) = |\boldsymbol{r}_{n,i} - \boldsymbol{r}_{n,1}|/(t_{n,i} - t_{n,1}) \tag{4.25}$$

is calculated, and the average $\langle v\rangle(i)$ over all particles n that had an ith collision is shown; again both the total and only the radial component of the velocities are plotted.

Finally, the right panel shows how information is travelling outwards. The outer boundary of its spreading is after a very fast expansion in the first few tenth of a fm/c travelling with approximately $0.17c$. The maximum distance of a scattering event from the beam axis is 2.5 fm. A more detailed analysis of the overlay of the average c.m.-energy and the distance of scattering events from the beam axis shows that the expansion of a shock-wave begins rapidly immediately after the c.m.-energies get smaller, but then is damped out – only the initial stages of the collision seem initial outburst remains.

In fact this model provides a way to address the problem of the energy-dependent parton resolution described in the beginning of Sec. 4.4: Disregarding interactions below a certain c.m.-energy cutoff could also be viewed as disregarding the interaction partners. By smearing out the step-function of the cutoff to a smoother function that determines the probability of scattering events in dependence of the c.m.-energy, one could try to simulate the behavior of the parton distribution function in dependence of the resolution parameter Q^2. However, other physics would have to take the place of the QCD interactions, in this lower-energy regime nucleonic interactions take over and hadronization patterns must be applied.

4.6.3 Model C: First Steps Towards a Retarded Interaction

As already pointed out, the main reason for the superluminous signal velocities is the instantenous character of the elementary interactions. To consequently overcome this problem no theoretical framework with time delays has been developed so far for parton systems, although some simple hadronic examples have been worked out. To estimate the effects of retardation it nevertheless seems appropriate to introduce a simple – but unphysical – way of time delay in the interactions: After an interaction took place the momentum transfer to both partons involved is delayed by $|\Delta \boldsymbol{r}_{\text{lab}}|/(2c)$, where $\Delta \boldsymbol{r}_{\text{lab}}$ is the distance vector in the lab frame. In the meantime the partons are considered not being able to interact, see Figure 4.14. A technical problem in the calculations however is that the time delay has to be implemented for integer numbers of timestep lengths and therefore does not necessarily fully correspond to its supposed length.

It turns out that the propagation of the outgoing shock-wave becomes more homogeneous, the initial outburst is suppressed while the distances of the outmost to be problematic. However, one should note that in the later stages the individual collisions happen with very high signal velocities, and that it is rather by the random-walk character of the signal propagation over longer distances that the information does not travel with the same high velocity, compare Sec. 4.2.

4.6.1 Model A: Restrictions on the Signal Velocity

An obvious way to avoid superluminous signal velocities is to systematically suppress all scattering events that would lead to signal velocities faster than the speed of light, the second row of Figures 4.12,4.13 summarizes the results of this simulation.

It is not surprising that the velocity restriction leads to a huge reduction in the collision rate: While in the first stages of the collision the collision rate drops to fourtynine percent of the original rate, it drops in the later stages to only about eight percent of the original rate, there is no outwards travelling information at all – it would be unwise to apply such an unjustified prescription given the huge effect, even more so, as from the extrapolation of intermediate energy results one would expect a rather strong outgoing effect of the proton impact.

4.6.2 Model B: Suppressing of Low Energy Collisions

Once a parton has scattered with a parton from the other nucleus it can scatter with fellow partons from the same nucleus at much lower c.m.-energies. Soft processes between partons from the same nucleus are problematic since the elastic gluon-gluon cross sections at low energies are large; systematically suppressing these reactions by placing a c.m.-energy cut-off on the scattering events should also suppress high signal velocities. For this model calculation the cross sections are set to zero below 0.4 GeV, see the third row of Figures 4.12,4.13. The outcome of this simulation proves that indeed the shock-wave travels outwards through those low energy collisions, only the

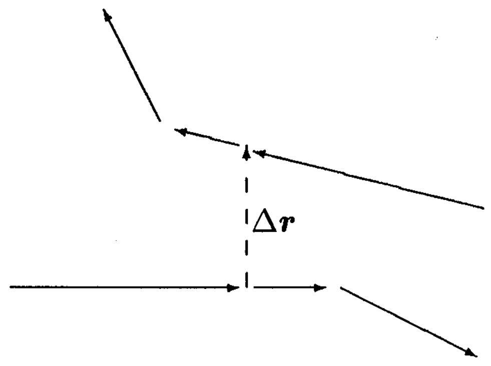


Figure 4.14: A first step to the introduction of retarded interactions.

scattering events are comparable to the simulations with instantaneous interactions, see the forth row of Figures 4.12,4.13.

4.6.4 Model D: Downscaling the Cross Sections

The maximum signal velocity is proportional to the maximum cross section. Downscaling the cross sections to $1/n$th of its original value and compensating this by the initialization of n times as many testpartons leads to a reduction of the maximum signal velocity by $1/\sqrt{n}$. Due to limitations given by the increasing computation time, for this model calculation a factor of only $n = 5$ has been chosen which corresponds to a reduction of the maximum signal velocity by a factor of ≈ 0.45.

It turns out that this technique radically changes the characteristics of the simulation, the outgoing shock-wave is strongly suppressed, see the fifth row of Figures 4.12,4.13. This outcome is surprising, since Ref. [Wel89] found that for internuclear cascade codes the outcome of this "full-ensemble" method (n times as many test particles) is comparable the usual "parallel-ensemble" method (n parallel test runs). In the case of parton cascades apparently this does not hold true anymore due to the reduction of superluminous signals.

Before favoring a method however that leads to such strong changes in the outcome of the simulation, further analysis is mandatory.

4.6.5 Model E: Wee-Partons

When initializing the parton configuration the Lorentz-contraction leads to very thin pancakes of nuclear matter, the contracted thickness of a nucleon is around $8 \cdot 10^{-3}$ fm, that of the gold nucleus $6 \cdot 10^{-2}$ fm. Thereby the z-coordinate of the partons is fixed rather precisely. However, the momentum of the low-x-components of the nucleonic wave-function is also determined within the order of a few hundred MeV. This leads to a violation of the uncertainty principle, which lead to the suggestion that in any frame of reference the nucleonic wave-function should be smeared out to a pancake thickness of at least approximately 1 fm, where the low-x-components are situated further outside and only the valence-quarks are actually within the highly contracted pancake [Gei92.1, Gei94.1, Gei92.2, Gei93, Gei94.2, Mcl82]. Applying this kind of initial configuration naturally leads to a smaller parton density; in this case approximately by a factor of 20. Figure 4.15 shows the parton configuration both at the beginning of the simulation and at 2 fm/c.

One's hope can be that the increased width and smaller parton density of the nuclei compared to the fully Lorentz-contracted model helps to eliminate some causality violating effects, see the bottom row of Figures 4.12,4.13: The initial outburst of a shock-wave at impact is eliminated because the nuclei enter each other rather gradually, an outgoing shock-wave within the respective nuclei is stronger damped because of a decrease in collision rate, and finally time-ordering problems on the scale of the time the nuclei are passing each other are rarified because that time-scale becomes much longer.

However, there are now other inconsistencies: Although this concept given by the

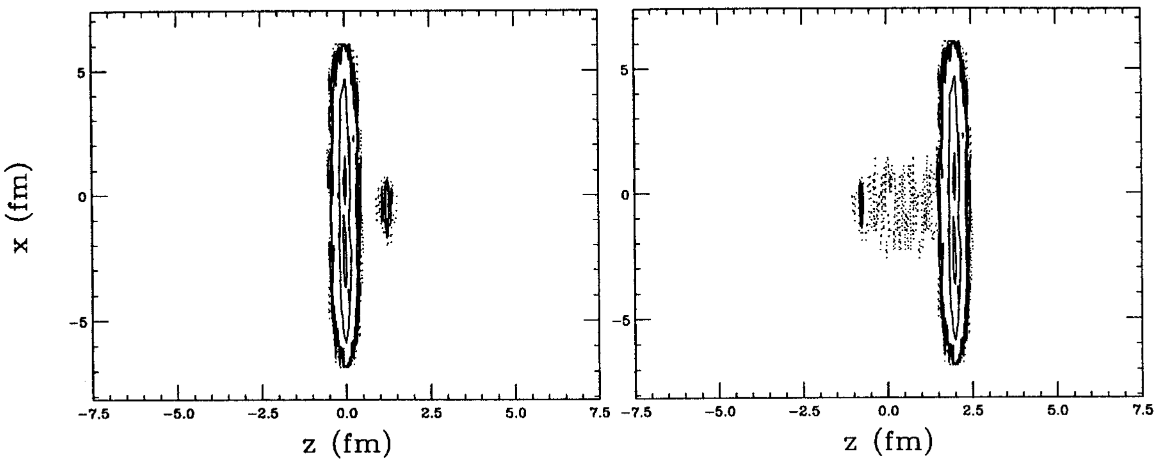


Figure 4.15: Initial configuration and configuration after 2 fm/c for a simulation with wee-partons.

uncertainty relation is truly valid in any frame of reference, within this model there is no way of implementing it in a Lorentz-invariant way. In reality the components of the wave-functions transform, in different frames different components of the wave-function are considered a parton. In the model used only the parton coordinates and momenta transform, while the partons themselves, once generated, exist in any frame. While for the initial collisions the c.m.-frames of individual parton collisions nearly coincides with the lab-frame, in subsequent collisions the c.m.-frames are closer to the rest-frames of the respective nuclei – in its rest-frame however, a nucleon, smeared out to 1 fm in the lab-frame, has a longitudinal radius of 100 fm.

4.6.6 Model F: Proper time approach

As already pointed out, a fully covariant description of a particle collisions would require a fully 4-dimensional configuration space. This is not possible without giving up the equal-time character of the simulation. Ref. [Kod84] however proposes a causality preserving scheme that while retaining the unique global time character of the simulation minimizes the frame-dependence of the choice of collision partners for the particles, but not the frame-dependent time-ordering of those collisions. Each particle i is considered to have its own clock showing its proper time τ_i,

$$d\tau_i = dt\sqrt{1-\beta_i^2(t)}, \quad \tau_i(t) = \int_0^t dt'\sqrt{1-\beta_i^2(t')} \ . \tag{4.26}$$

Since the particles do not change their momenta between collisions, the above integral can be reduced to a sum of products of the type $\beta_{i\alpha}\Delta t_\alpha$. With $\tau_{ic}(j)$ being the proper i-time of the collision of particle i with particle j, and τ_{i0} being the proper i-time of the most recent collision of particle i, let

$$\delta\tau_i(j) := \tau_{ic}(j) - \tau_{i0} \tag{4.27}$$

be the proper time distance between those two events, a Lorentz-invariant quantity. The collision instant $\tau_{ic}(j)$ is defined individually within the rest-frame of particle i through the closest approach to particle j – in the other methods, the closest approach is defined either within one frame, usually the lab-frame or the c.m.-frame of the nuclei, or within the respective c.m.-frame of a particle *pair*. In the previous discussions, the latter mechanism had been chosen. To co-relate the individual closest approach tests, for collisions only the particle pairs (i,j) are considered for which both

$$\delta\tau_i(j) = \mathrm{Min}\,\{\delta\tau_i(l) > 0,\ l = 1,\ldots,N; l \neq i\} \tag{4.28}$$

and

$$\delta\tau_j(i) = \mathrm{Min}\,\{\delta\tau_j(l) > 0,\ l = 1,\ldots,N; l \neq j\} \ , \tag{4.29}$$

N being the total number of particles in the simulation. Through the restriction $\delta\tau > 0$ only collisions in the absolute future of each particle are considered. What the two above conditions mean is that for particle i the very next possible collision (in the sense of proper time) is with particle j *and vice versa*. This algorithm only allows collisions that have no risk of not happening in another frame. Suppose particle 1 has a minimum proper time distance to particle 2, but particle 2 has its minimum proper time distance to particle 3, then no collision will take place at all.

The search for the very next collision partner of every particle in the simulation is an un-avoidable N^2-problem. Since each of these tests involves a change of coordinate system, this mechanism is computationally very intense. Therefore, it was only possible to simulate the initial stages of one single (p,Au)-collision. It was observed that the number of collisions in this phase dropped dramatically, in this testrun to about 10% of the number in the other simulations; due to computational limitations however one is not able to claim significant statistics on this percentage. This outcome is compatible with the conjecture of the authors of Ref. [Kod84] that the mechanism might underestimate the number of collisions [Kod94]. In smaller simulations of (p,p)-collisions situations were observed where particle 1 had particle 2 as its closest collision partner, particle 2 had particle 3, and particle 3 again had particle 1. "Ring"-configurations like this, possibly spanning over even more particles, might lead to the underestimation of the collision rate. This effect might have been enhanced by the high particle density and the extreme Lorentz-contraction of the nucleons as compared to lower energy internuclear cascades the mechanism was developed for. Further investigation of this method is required, but due to computational limitations was not possible within this work. It is however not expected that the above mechanism can overcome the macroscopic problem of shock-waves.

4.7 Conclusions

The influence of superluminous signal velocities on the signal propagation in parton cascade codes was found to be smaller than expected. This is mainly due to two effects: In the initial stages of the interaction the energy-dependent cross sections tend to be small. Without this beneficial effect the signal propagation has a threshold in the region between 0.4 and 0.5 fm^2 from where on the velocity of the outgoing shock-wave reaches the speed of light. In the later stages of the interaction shock-waves get damped out not because of a lack of interactions but because of signal propagation that resembles a random walk: not every collision actually leads to an outward propagation, in fact, the equally probable inward propagation leads to a virtual damping of the shock-wave.

APPENDIX

Appendix A

The Parton Cascade Program

A.1 Flow chart

After initialization of the two pancakes by placing the test particles into the configuration space and setting their initial momenta, the program mainly repeats the steps

- Moving the particles according to their momenta
- Checking for collisions between the particles.
- Decaying of particles.

This is done for a given amount of timesteps.

The program structure is shown in the flow chart Figure A.1.

A.2 Test runs

The program is able to simulate the behaviour of more than one particle system at a time. Each of the particle systems is called a 'test run'. Since scattering is done by Monte Carlo methods each of the configurations which are initially similar will behave differently on the long run. The test runs however are set up to start with slightly different initial conditions, too.

The test runs can be used to find averaged properties of the systems. They can also be used to calculate mean fields for the particles, but this is not currently implemented. As long as no mean field is taken into account, the test runs are independent from each other.

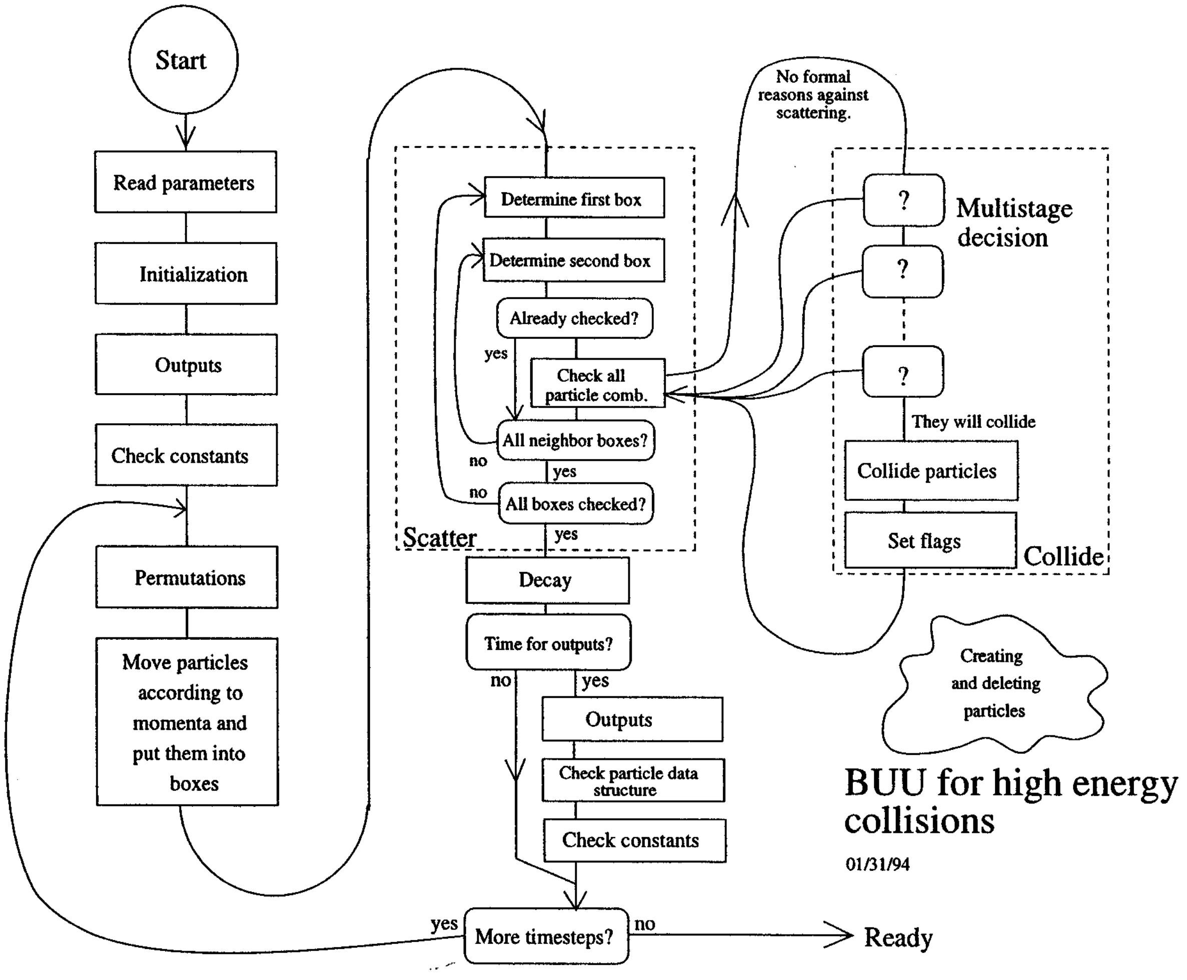


Figure A.1: BUU program flow chart

A.3 Scattering

As already explained running the simulation consists mainly of two steps: Moving the particles according to their momenta as one step, scattering the particles as another step. An additional step can be the decaying of partons. In principle the scattering step involves the examination whether or not scattering takes place for every particle pair in the test runs. This would result in an N^2-time dependence of this step. However since the spatial range of the interactions taken into consideration is limited, most of the time particle pairs that will never have a chance of being scattered would be examined.

To overcome this waste of time the coordinate space is divided into boxes constructed in a way such that only particle combinations with members of the same or neighboring boxes have to be taken into account. The boxes establish a grid in configuration space.

This implies that initialy in the x and y direction the edges of the boxes have to be longer than $2\sqrt{\frac{\sigma_{\text{max}}}{\pi}}$ with σ_{max} being the maximum cross section taken into account. In the z direction the edges have to be longer than $2t_{\text{step}} \cdot c$ with t_{step} being the timestep length. During program execution the z extension has to be adjusted as particles are scattered from their original z or $-z$ direction respectively. With θ_{max} being the maximum angle any one particle has from the initial direction, the grid extension in z direction is adjusted to $2t_{\text{step}} \cdot c + 2\sqrt{\frac{\sigma_{\text{max}}}{\pi}} \sin(\theta_{\text{max}})$.

The total extension of the grid has to be chosen such that during the whole simulation no particle leaves the grid, the latter has to be checked during the particles are moved. The assignment of the particles to the boxes is also done during the move-step.

In the scattering step all box combinations between neighboring boxes have to be selected, and for each box combination all particle combinations have to be examined. This is done in the following way:

To prevent the occurence of preferential directions always first one box is chosen at random, then in a random order one of the neighboring boxes or itself is chosen as a second box. It is made sure that each box combination is only taken into account once. Then all particle combinations with one particle from the first box and the other one from the second box are checked.

Before the physics of a possible scattering is checked, a few formal knock out criteria are applied:

- Neither of the particles has already scattered in this timestep. This is done to avoid possible violations of causality: One timestep length is considered the smallest scale on which any one thing can happen to a particle.

- Not both of the particles have had the respective other one as their last scattering partner. This is done to avoid multiple scattering, a phenomenon already taken into account on the level of elementary processes.

- Not both of the particles haven't scattered with a particle from the respective other nucleus of origin before. This is done to prevent scattering between particles from the same nucleus before any collision with the other nucleus had taken place – since no mean field guarantees the stability of an unperturbed nucleus, stability is provided in this artificial way by the lack of any interaction. In principle the same effect can be achieved by having a cutoff energy that suppresses interactions with a too small c.m.-energy.

A.4 Assignment of the particles to the boxes

Since during the scattering process always first the boxes are chosen, it is necessary that, given a box number, all particles in that box can be accessed rapidly. However, since the particle data is stored independently of any box membership, assignment to the boxes is rather done by a pointer structure being established each time new when the particles are moved.

This movement of the particles is done in a random order; each box possesses one pointer that points to the first particle being found in the box area after moving, or to nowhere if the box area turned out to be empty.

Each particle itself possesses a pointer that points to the next particle being found in the same box, or to nowhere if it was the last particle being found there.

As a result all particles in a certain box can be accessed by following this chain of pointers, starting with the boxes' own pointer and continuing through the particles' 'next-particle'-pointer, until any of those points to nowhere.

A.5 Checking for collisions

After two particles have been admitted for scattering by formal reasons (see A.3 on page 81) the physics criteria for scattering or not are checked, in particular:

- Are the two particles close enough for the particular c.m.-energy dependent total cross section of the reaction?
- Have they reached their point of closest approach in the c.m.-system?

While the first condition can be checked very quickly, performing the other two tests takes much more computation time since it involves a transformation into the c.m.-system of the two particles.

To avoid this as long as possible, a multistage decision is applied: Weaker criteria which take less computation time are checked first, the decision sequence is immediately aborted as soon as one of its requirements is not met.

A.6 Decay of particles

Off-shell particles that don't belong to a certain nucleon anymore and haven't collided with another particle in the actual timestep are assumed to be able to decay. Due to the uncertainty relation

$$\Delta E \cdot \Delta t \sim 1$$

their lifetime t_{life} is to be chosen according to

$$\Delta m \cdot t_{\text{life}} \sim 1 \; .$$

Δm is the off-shellness of the particle and calculated according to

$$\Delta m = \left|\sqrt{p_\mu p^\mu} - m_{\text{on-shell}}\right| \; .$$

$m_{\text{on-shell}}$ is the on-shell mass of the particle, p_μ its four momentum.

In order to find the probability p for them to decay in the actual timestep the livetime t_{live} is assumed to be the expectation value of possible lifetimes:

$$t_{\text{live}} \overset{!}{=} \frac{\sum\limits_{n=0}^{\infty} p_n n t_{\text{step}}}{\sum\limits_{n=0}^{\infty} p_n}$$

Here p_n is the probability for the particle to survive n timesteps with a total lifetime of nt_{step}. This must be equal to the probabilty of being not "killed" in n timesteps:

$$p_n = (1-p)^n$$

Therefore

$$t_{\text{life}} = t_{\text{step}} \frac{\sum (1-p)^n n}{\sum (1-p)^n}$$

and with

$$\sum x^n = \frac{1}{1-x}, \quad \frac{\partial}{\partial x} \Longrightarrow \sum x^n n = \frac{x}{(1-x)^2} \qquad |x| < 1$$

finally

$$t_{\text{live}} = t_{\text{step}} \frac{1-p}{p}$$

and

$$p = \frac{\Delta m t_{\text{step}}}{1 + \Delta m t_{\text{step}}} \; .$$

In the decay step for all particles in question, a random number between 0 and 1 is determined and compared to p. If it is smaller than p the particle decays into two other particles.

A.7 Creating and deleting particles

Since during execution oftentimes particles are created or deleted, an efficient method had to be found for this.

As all particle information is stored in static fixed-size arrays, the simplest way would be to always have those arrays filled up to a variable index. This maximum index would be stored elsewhere. Whenever a new particle is created, it would be placed on top of the others and the maximum index would be increased, whenever a particle is deleted, the stack above it would have to fall down and fill the gap, and the maximum index would have to be decreased by one. However the "falling down" involves a lot of copying, also, a lot of pointer information would have to be updated.

The other extreme would be not to care about maximum indices and instead only to provide a flag for each array entry whether or not it is used. When a new particle is to be created one would use some free entry in the array, write in the information and flag it 'used'. Deleting a particle would simply be done by marking the corresponding entry 'unused'. No pointer structures would be updated, one would always run through the whole array or pointer chains and just skip 'deleted' particles. However, if the arrays are only filled rarely, this would be a waste of time.

So a combination of both methods is chosen:

- The array entries can be marked 'unused'.
- The maximum used index number is stored elsewhere, all loops only have to run up to this index.
- The pointer chain for the box contents is updated.

When a particle is created, beginning from the bottom of the array the first unused position is searched for and the information written to it. The entry will be marked 'used'. If it is above the maximum index so far (which can only be by '1') the maximum index is updated. The new particle is not incorporated into any pointer structures since as it already had interacted it would not take part in any other interaction in this timestep anyway.

When a particle is deleted its array position is marked 'unused'. The particle is removed from the box pointer chains, the maximum array index is updated.

Appendix B

Program listing

This is not a complete listing of the program, but only a selection of the most relevant subroutines and files.

B.1 STDEC

Gerd Kortemeyer

Last revision of file: 08/31/94

This is the include file with all declarations of the common blocks used for communication between the subroutines.

It also sets

```
c Type check on
      implicit none
c
```

which forces the programmer to declare any local variables explicitly. Some physics parameters are also set.

The standard declarations are to be included in any subroutine that is supposed to manipulate global data. This is done simply by adding

```
          include 'stdec.for'
```

Parts of this file should not be copied directly into subroutines since this might result in a lot of confusion if at some stage the definition of common-blocks is changed in the standard declarations file. The compiler will not be able to figure out that the definition of some common block is different in different subroutines and map the variables wrong.

B.1.1 Physics parameters

The include file defines the following constants:

```
c Mass of nucleon [GeV/c**2]
      double precision manuc
      parameter (manuc=0.939)
c hbar*c [GeV*fm]
      double precision hbc
      parameter (hbc=0.197327)
c r0 [fm]
      double precision r0
      parameter (r0=1.2)
c
```

B.1.2 Other parameters

```
c
c Parameters for the dimensions of arrays:
c
c Maximum number of quarks
      integer    mquark
      parameter (mquark=15000)
c Maximum number of test runs
      integer    mtest
      parameter (mtest=5)
c Grid-dimensions
      integer mgridxy,mgridz
      parameter (mgridxy=20)
      parameter (mgridz =250)
c Dimensions for cross section array
      integer maxpoints
      parameter (maxpoints=3000)
      double precision sqstep
      parameter (sqstep=1000./(maxpoints-1.)**4)
c Maximum particle type index
      integer maxtype
      parameter (maxtype=6)
c Maximum number of channels
      integer maxchannel
      parameter (maxchannel=6)
c Very small number
      double precision eps
      parameter (eps=.0000000001)
c
c The number PI
c
      double precision pi
      parameter (pi=3.1416)
```

B.1.3 Particle data

In the common block `particles` all information that goes with each of the single particles is collected.

The particles are identified by two numbers: The number of the test run they are in, ranging from 1 to `mtest`, and the particle number in that run, ranging from 1 to `mquark`.

As the arrays are dimensioned staticly the two variables `ntr` and `nq(mtest)` are included. `ntr` (int) holds the number of test runs really active, it is set in subroutine `parmin` from the information in the parameter file.

`nq(mtest)` (int) gives the maximum particle-index per test run. It is first set in subroutine `init`, but can vary when new particles are produced or other particles are absorbed. This is done in the procedures `crepar` (see B.12.2, p. 152) and `delpar` (see B.12.1, p. 149).

So if one wants to construct a loop over all particles a typical structure would be

```
do 10 test=1,ntr
   do 20 particle=1,nq(test)
```

This doesn't necessarily mean that all array entries between 1 and `nq(test)` are actually used.

The particle information includes

- `tq(mtest,mquark)` (int), the parton type. This is 0 for gluons, 1 for up-quarks, 2 for down-quarks, 3 for strange-quarks, 4 for charm-quarks, 5 for bottom-quarks and 6 for top-quarks. The accompaining anti-quarks have got the same but negative number, so for example -1 stands for anti-up. This parton code corresponds to the codes defined in the CTEQ-routines for parton-distributions.
- `rq(mtest,mquark,3)` (double), the position of the particles. The last array index corresponds to the x, y and z components of the position vector respectively. The unit is fm. The time component of the position is defined by the time step.
- `pq(mtest,mquark,3)` (double), the momentum of the particles. Again the last array index corresponds to the spatial components of this vector. The unit of the spatial components is GeV/c. The energy component is given separately by the array `eq(mtest,mquark)` (double), its unit is GeV.
- `nnq(mtest,mquark)` (int), the number of the nucleon the particle was assigned to during the initialization. The total number of nucleons is given by `nntar` and `nptar` for the number of neutrons and protons in the target, and by `nnpro` and `nppro` for the number of neutrons and protons in the projectile. The numbers `nntar` etc. are set in `parmin` by the user, `nnq` is set in `init`.

  A particle that was scattered by subroutine `collide` will have `nnq`=0 afterwards, a particle created by subroutine `crepar` will have `nnq`=-1.
- `fq(mtest,mquark)` (int) is a flag wether the particle belongs to the projectile (`fq`=-1) or to the target (`fq`=1), they are first set in subroutine `init`. This flag is used in subroutine `scatter` to prevent particles from the same nucleus to scatter with each other before at least one of them has scattered with a particle from the other nucleus. After their first scattering with any particle their flag is set to `fq`=0. A particle with `fq`=0 can then scatter with any other particle.
- `last(mtest,mquark)` (int) is a pointer to the particle in the same test run the actual particle has last scattered with, so values for `last` naturally range from 1 to `mtest`.

  But `last` has a second usage: `last`=-1 means that the particle has been deleted in the last interaction. So `last` also flags empty array positions.

  However particles that haven't scattered before carry `last(i,j)`=j as preset by the subroutines `init` or `crepar`.

  So to write a loop over all *existing* particles one should code

  ```
  do 10 test=1,ntr
     do 20 particle=1,nq(test)
        if (last(test,particle).ne.-1) then
           ...
  ```

The values for last are set in subroutine collide and used in subroutine scatter to prevent particles from being engaged in multiple scattering with each other. Two subsequent scattering events between the same particles are forbidden, however it is possible to scatter particles 1 and 2, then 2 scatter with 3, finally 1 again with 2. This results in a kind of 3-body-force.

- nextgrid(mtest,mquark) (int) points to another particle of the same test run that is in the same space box as itself. If it is the last particle in that chain the value for nextgrid is -1. The procedure is decribed in the section for procedure move, B.5 on page 114.
- lastpoint(mtest,mquark,3) (double) The most recent point in the lab frame that the parton scattered at. The scattering point is assumed in the middle between both partons involved.
- lasttime(mtest,mquark) (int) The most recent timestep-number that the parton scattered in. Both lastpoint and lasttime are needed to calculate the signal velocity from one scattering event to the next.
- fdue(mtest,mquark) (int), feq(mtest,mquark) (double), fpq(mtest,mquark,3) (double) This is data for experiments with retarded interactions, fdue is the amount of timesteps that the parton is considered to be involved in the interaction, feq and fpq are the energy and the momentum that the parton is supposed to get.

B.1.4 Other parameters

In the common block others control parameters for the program run are collected.

In particular there are

- ntime (int) The actual number of the timestep being executed at the moment.
- timstp (double) The length of one timestep in fm/c. This value is read from the parameters file.
- prox0 (double) The x coordinate in fm of the projectiles starting position, defined in the parameters file.
- proz0 (double) The z coordinate in fm of the projectiles starting position, defined in the parameters file.
- iseed (double) The starting number for the random number generator, defined in the parameters file.
- nmts (int) The total number of timesteps to be performed, defined in the parameters file.
- whenout (int) The number of timesteps to be performed between two 'big' outputs, for example plots of the particle position in coordinate space, output of statistics.... Read from the parameters file.
- elabpnpro (double) The kinetic energy in GeV per nucleon of the projectile, defined in the parameters file.
- elabpntar (double) The kinetic energy in GeV per nucleon of the target, defined in the parameters file.
- maxsig (double) The maximum total cross section in fm^2 as being found during the initialization of the cross section tables xsec..., see subroutine initcross (B.3.7, p. 106).
- gridspxy (double) The x and y length in fm of the box edges. The value is determined according to the maximum cross section maxsig.

- `gridspz` (double) The z length in fm of the box edges. The value is re-determined after every timestep according to the maximum angle a parton is headed away from it's forward direction in connection with `maxsig` and the timestep length `timstp`. This is done in subroutine `move` (see B.5 on page 114):

  ```
  gridspz=2.*(timstp+sqrt(maxsig/pi)*sin(mtheta))
  ```

- `gridz` (double) is the bin-width for configuration plots.

The items `elabtar` and `elabpro` became obsolete and were removed with the introduction of a new initialization method for the particles. The item `lowcut` became obsolete with a new treatment of low energy collisions.

B.1.5 Grid information

In the common `gridinfo` information for the spatial grid is stored.

In the integer array

```
gridroot (mtest,-mgridxy:mgridxy,-mgridxy:mgridxy,-mgridz:mgridz)
```

the pointers to the first particles found in the boxes are stored. A '-1' stands for 'no particle', meaning 'the box is empty'.

The first index represents the test run number, the following indizes the box coordinate. In the x and y direction the boxes are numbered from `-mgridxy` to `mgridxy` with box number `-mgridxy` being the leftmost or lowest boxes respectively. The z coordinate is described by the last index.

The integer arrays

```
permx(-mgridxy:mgridxy),permy(-mgridxy:mgridxy),permz(-mgridz:mgridz)
```

are permutations of the numbers from `-mgridxy` to `mgridxy` or from `-mgridz` to `mgridz` respectively and are used to have access to the boxes in a random way.

Instead of accessing `gridroot(test,i,j,k)` during scattering rather

```
gridroot(test,permx(i),permy(j),permz(k))
```

is accessed. The random order of accessing the boxes is necessary because with each box also the neighboring boxes will be accessed and a fixed order for that would result in preferential directions.

The integer arrays

```
deltx(3),delty(3),deltz(3)
```

have the same function for accessing the neighboring boxes in a random way, they have a permutation of the numbers from -1 to 1.

B.1.6 Initialization parameters

In the common block `initparms` information concerning the different methods of initialization are stored as they are being read from the parameter file.

- `initmeth` (int) In this variable the number of the initialization method to be used is stored. It is read in `parmin` and used in `makenucleus` (see B.3.2 on page 103).
- `initxmin` (double) The minimum x value to be used in the parton distribution function $f(x,Q^2)$, see B.14 on page 157.
- `initqsq` (double) The Q^2 value to be used in the parton distribution function $f(x,Q^2)$, see B.14 on page 157.
- `initrnuc` (double) The nucleon ('bag') radius assumed.

B.1.7 Information block for the initialization methods

In the common block `create` all necessary information for the different initialization methods is stored It has to be filled before `makenucleus` is called. In particular:

- `crelabpn` (double) The kinetic energy per nucleon of that nucleus in the lab frame.
- `crex0` (double) The initial x position of the nucleus' center in the lab frame.
- `crez0` (double) The initial z position of the nucleus' center in the lab frame. The y position is chosen to be zero always.
- `crenp` (int) The number of protons in the nucleus.
- `crenn` (int) The number of neutrons in the nucleus.
- `cremodir` (double) The direction for the momentum: '1' means 'travelling in positive z direction', '-1' means 'travelling in negative z direction'.
- `creflag` (double) The flag the particles from this nucleus should have. This corresponds to the array `fq` in the particle data block.

B.1.8 Data for two particles that are scattering candidates

In the common block `twopart` gradually information about two particles that are defined by subroutine `scatter` (B.6, p. 118) for being candidates for scattering is collected.

The data is mainly set and used in subroutine `collide` (B.8, p. 123), but also in `twototwo` (B.9, p. 139).

In particular the entries are

- `tpp1` (int) The number of the first particle being chosen.
- `tpp2` (int) The number of the second particle being chosen.
- `tprn` (int) The number of the test run the particles are from.
- `tpsq` (double) The total energy squared in GeV^2 of the two particles in their c.m.-system.
- `tprl(3)` (double) The distance vector of the two particles in the lab frame in fm.
- `tpeq1, tpeq2` (double) The energies of the two particles in GeV in the c.m.-system.

- `tpmo1(3)` (double) The vector of momentum in the c.m.-system for the first particle in fm/c before scattering.
- `tpbeta` (double) The transformation vector $\vec{\beta}$ to enter the c.m.-system.
- `tpeql` (double) The total energy of the two particles in the lab frame.
- `tpbq` (double) The value of β^2.
- `tpgamma` (double) The value $\gamma = \frac{1}{\sqrt{1-\beta^2}}$.
- `tpsigma` (double) The total cross section of the two particles in fm^2 as being set by subroutine `totalcross` (B.8.3, p. `totalcross`). However, for technical reasons after `xsecentry` (B.8.4, p. 136) has been called, the variable includes the partial cross section for the specified process.
- `tpgambet` (double) The value of $\frac{\gamma-1}{\beta^2}$.
- `tptd` (double) The time difference between the two particles when seen in the c.m.-frame.
- `tprc(3)` (double) The spatial distance vector in the c.m.-frame.
- `tpcos` (double) The cosine of the scattering angle θ in the c.m.-frame.
- `tpsc1(3)` (double) The momentum vector of the first particle in the c.m.-system after scattering. It is produced from `tpmo1` by subroutine `newmomenta`.
- `tpsceq1, tpsceq2` (double) The energies of the two particles after scattering.
- `tpchin` (int) The channel number of the two incoming particles. The channels are coded in the following way:

Channel number	particle content
1	gg
2	qg
3	qq
4	$q\bar{q}$
5	qq'
6	$q'\bar{q}'$ – outgoing channel only

  The prime denotes a quark different from the unprimed quark, the bar denotes the respective antiparticle.

- `tpchout` (int) The channel number of the two outgoing particles.
- `tpcrindex` (int) The energy index for the arrays `xsec...`, see B.1.16 on page 93.
- `tpdq` (double) The squared distance in fm^2 between the two particles in the lab frame.
- `tpdqc` (double) The squared distance in fm^2 between the two particles in the c.m.-frame.
- `tin1, tin2` (int) The parton types of the two incoming partons as being set in `totalcross` (B.8.3, p. 135).
- `tout1, tout2` (int) The parton types of the two outgoing partons as being set in `outchannel` (B.8.2, p. 132).

The item `tpeql` became obsolete.

B.1.9 Already scattered

In the common block `already` there is only one array `alr(mtest,mquark)` (int) that flags for each particle if it had already been scattered in this timestep or not. '1' denotes 'scattered', '0' denotes 'not scattered'.

It is reset in subroutine `scatter`, set in `twototwo` and used in both `scatter` and `decay`.

B.1.10 Statistics

The common block `stat` includes some statistics information that is being printed out to the protocol file during 'output'.

- `minsigma, maxsigma` (double) The minimum and maximum values in fm^2 for the cross sections occurred between the outputs.
- `minsq, maxsq` (double) The minimum and maximum values for the c.m.-energy squared in GeV^2 occurred between the outputs.
- `mincos, maxcos` (double) The minimum and maximum cosine of the scattering angle in the c.m.-frame as determined by the montecarlos.
- `nsta0(mtest),..., nsta4(mtest)` (int) Counters for the multistage check: They are incremented each time one particle combination passes the first, second,... stage of the test. This can be used to supervise the efficiency of the test. For each testrun there exists a separate counter.
- `timefault` (int) This counter is incremented each time the time-distance between two particles that occurres during transformation to the c.m.-system is bigger than the transformed timestep length.
- `anglestat(maxchannel,6)` (int) With this array the c.m. angular distribution of scattering events in the respective output channels is examined. The second index belongs to bins with $\cos\theta$ from -1 to $-\frac{2}{3}$, $-\frac{2}{3}$ to $-\frac{1}{3}$, ... The contents of this array are written to the protocol file at every 'big' output.
- `labangle(6)` (int) With this array the lab angular distribution of scattering events is examined. The index belongs to bins with $\cos\theta$ from -1 to $-\frac{2}{3}$, $-\frac{2}{3}$ to $-\frac{1}{3}$, ... The contents of this array are written to the protocol file at every 'big' output.
- `chscat(maxchannel,maxchannel` (int) In this array the scattering events from the channel number in the first index to the channel number in the second index are counted.

It also includes information about the number of particles created or deleted:

- `crenumber, delnumber` (int) The number of particles created/deleted since the last output.
- `lastnumber` (int) The number of particles at the previous output, used for checks.

B.1.11 Information for particles to be created

The common block `newpart` has to be used when a new particle is to be created by subroutine `crepar`, see B.12.2 on page 152.

In particular the following information has to be provided: The position `newrq(3)`, the momentum `newpq(3)`, the energy `neweq`, the parton type `newtq`, and the testrun `newtest` that the particle should be created in.

The item `newpscat` became obsolete with the removal of the array `pointscat`.

B.1.12 Permutation

The common block `perm` only has one entry, the array `perm(mquark)` (int). It is initialized to hold all numbers from 1 to `mquark`, but in a permuted way. This array can be used to access the particles in a random way by first building a subpermutation with the appropriate number of particles and then accessing the particles always through the subpermutation, see B.5.3 on page 116.

B.1.13 Timing information

The common block `time` has the variable `t0` (real) that can be used to find the total CPU-time so far by calling `secnds(t0)`.

B.1.14 Controls

The common block `control` has integer entries determining wether or not certain outputs are written. '0' means 'no', '1' means 'yes'. The values are read from the parameters file. In particular these are

- `ploto` (int) Wether or not the configuration and rapidity plots are done.
- `scattering` (int) Wether or not information about scattering events is written out, see subroutine `twototwo` on page 139.

Both the items `causalpl` and `velocitypl` became obsolete with the introduction of `scattering`.

B.1.15 Charges

In the common block `charges` there is one array `charge` that has the charge in units of the electron charge for every parton type.

B.1.16 Cross sections

In these arrays the partial cross sections for different processes are stored. The entries are of type `double precision`:

```
xsecgg22(maxpoints),
xsecqq22(maxtype,maxpoints),
xsecqg22(maxtype,maxpoints),
xsecqqb22(maxtype,maxpoints),
xsecqqp22(maxtype,maxtype,maxpoints),
xsecqqbgg(maxtype,maxpoints),
xsecggqqb(maxtype,maxpoints),
xsecqqbqpqbp(maxtype,maxtype,maxpoints)
```

The last index always refers to the energy mesh the cross sections are tabulated on. The preceding indices refer to the parton types involved. The channel combination is coded in the name of the arrays. For a description of these cross section arrays see B.3.7 on page 106.

For the incoming channel gg the open output channels are gg or $q\bar{q}$. The partial cross sections for two incoming gluons are added up in the order $gg \to gg$ (`xsecgg22`), $gg \to u\bar{u}$, $gg \to d\bar{d}$, $\ldots$, $gg \to t\bar{t}$ (`xsecggqqb`).

For the incoming channels qg and qq only the elastic channel is open.

For the incoming channel $q\bar{q}$ the partial cross sections are added up in the following order: $q\bar{q} \to q\bar{q}$ (xsecqqb22), $q\bar{q} \to gg$ (xsecqqbgg), $q\bar{q} \to q'\bar{q}'$ (xsecqqbqpqbp). The cross section array entries for $q\bar{q} \to q'\bar{q}'$ are added up in the order $q\bar{q} \to u\bar{u}$, $q\bar{q} \to d\bar{d}$ and set to zero when $q = q'$.

As one can see, the partial cross sections always add up to the total cross section in the last entry of that sequence, the difference between the entries and therefore the relative probabilty for the channel are the partial cross sections themselves. For the initialization of the `xsec...` arrays see B.3.7 on page 106.

B.1.17 Parameters for the cross sections

The following parameters are read from the parameters file and used in the cross section and montecarlo routines:

- `crm` (double) is the cutoff for the gluon propagator in the t and u channels.
- `crn` (double) is the cutoff for the gluon propagator in the s channels
- `crk` (double) is the cutoff for the quark propagators in any channels
- `cralphas` is the value for α_S.

B.1.18 Causality information

The common block `causal` with the items `steps`, `stymax`, `stcount`, `veymax`, `vc(0:mvelocity)` and `vgrid` became obsolete.

B.2 The main file

Gerd Kortemeyer

Last revision of the file: 04/27/94

This file consists the main program and includes the other major parts.

B.2.1 Local variables

```
c Date/Time
      character dat*9,tim*9
c Output-Counter, do variables
      integer outp, i, j
c For initialization of random number generator
      double precision dummy
```

B.2.2 Some initializations

First, date and time are read out and the timer for the CPU-seconds is reset

```
c Get date/time
      call date(dat)
      call time(tim)
c Reset timer
      t0=secnds(0.0)
```

Next, the protocol file is opened:

```
c
c File for all control outputs
c
      open(1,file='buu_protocol.dat',status='unknown')
c
      write(1,*) '               *** BUU for high energy collisions ***'
      write(1,*) ' '
      write(1,*) 'National Superconducting Cyclotron Laboratory'
      write(1,*) 'G.K. 1993, 94'
      write(1,*) 'Date: ',dat,' Time: ',tim
      write(1,*)
     & '============================================================='
```

The parameters are read in from the parameter file:

```
c
c Initialization of commons
c
      call parmin()
```

Now, the random number generator is initialized with the `iseed` given in the parameter file. The random number generator is designed to start a new sequence of numbers when a negative value is given as the argument:

```
c
c Reset random numbers
c
      iseed=-abs(iseed)
      dummy=ran1(iseed)
```

The particle data and some other data is initialized by subroutine `init`.

```
c
c Initialization of particle data
c
      call init()
```

The arrays for the causality and velocity plots are reset:

```
c
      steps=int(nmts/50.)
      stcount=0
      stymax=0
c Reset causal information
```

```
      do 100 i=0,mgridxy
         nc(i)=0
100   continue
c
      vgrid=mvelocity*timstp/sqrt(maxsig/pi)
c
      veymax=0
c Reset velocity information
      do 110 i=0,mvelocity
         vc(i)=0
110   continue
```

The graphics output files belonging to the output options given in the parameters file are opened:

```
c
c File for graphics output
c
      if (ploto.ne.0) then
         open(3,file='buu_plots.plt',status='unknown')
         write(3,*) 'SET STORAGE X Y Z SIZE=1000000'
      endif
c
      if (causalpl.ne.0) then
         open(4,file='buu_causal.plt',status='unknown')
         write(4,*) 'NEW FRAME'
         write(4,*)
     & 'TITLE 1 9.7 SIZE 2 ''HIGH ENERGY BUU ',dat,' ',tim,'CAUSALITY'''
         write(4,*) 'CASE '' LLL  LLLLL                    ',
     &'         LLLLLLLL'''
         write(4,*) 'SET FONT DUPLEX'
         write(4,*) 'SET LABELS SIZE=2'
         write(4,*) 'READ MESH'
         write(4,*) 'FOR Y=',(i*gridspxy,i=0,mgridxy)
      endif
c
      if (velocitypl.ne.0) then
         open(5,file='buu_velocity.plt',status='unknown')
         write(5,*) 'NEW FRAME'
         write(5,*)
     & 'TITLE 1 9.7 SIZE 2 ''HIGH ENERGY BUU ',dat,' ',tim,'VELOCITY'''
         write(5,*) 'CASE '' LLL  LLLLL                    ',
     &'         LLLLLLL'''
         write(5,*) 'SET FONT DUPLEX'
         write(5,*) 'SET LABELS SIZE=2'
         write(5,*) 'READ MESH'
         write(5,*) 'FOR Y=',(i/vgrid,i=0,mvelocity)
      endif
```

The procedure `plotout` is called to plot a picture of the initial particle distribution in configuration space and the initial rapidity distribution:

```
c
      if (ploto.ne.0) call plotout()
```

The procedure `constant` calculates some constants of motion and writes them to the protocol file:

```
c
      call constant()
```

Now the big loop over all timesteps starts. The counter `outp` counts the timesteps since the last 'big' output, the parameter `whenout` determines after how many steps output is done.

```
c
c Loop for all timesteps
c
      outp=1
c
      write(*,*) 'Starting: ',dat,', ',tim
c
      do 10 ntime=1,nmts
c
```

This is the kernal of the loop: Calling `move`, `scatter` and `decay`. However, since $1 \rightarrow 2$ parton processes are not yet implemented, the corresponding call statement is commented out.

```
         call permute()
         call move()
         call scatter()
*         call decay()
```

Now it is checked if it is time for 'big' output:

```
c Graphics and other output?
         if (outp.eq.whenout) then
c
            call date(dat)
            call time(tim)
c
            write(*,*) 'Timestep',ntime,' of',nmts,': ',dat,', ',tim
c
            write(1,*)
     &'---------------------------------------------------------------'
            write(1,*)
     &'---------------------------------------------------------------'
            write(1,*) 'Timestep:',ntime
            write(1,*) 'Date: ',dat,' Time: ',tim,' CPU:',secnds(t0)
            write(1,*) '-'
            if (ploto.ne.0) call plotout()
            write(1,*) '-'
            write(1,*) 'Multistage collision check statistics:'
            do 20 i=1,ntr
               write(1,*) 'Multistage: ',
     &              nsta0(i),nsta1(i),nsta2(i),nsta3(i),nsta4(i)
20          continue
            write(1,*) '-'
            write(1,*) 'Channel combinations:'
            do 30 i=1,maxchannel
```

```
              write(1,*) (chscat(i,j),j=1,maxchannel)
30         continue
           write(1,*) 'Angular distributions, cos(theta) in c.m.frame:'
           do 40 i=1,maxchannel
              write(1,*) '--------------------------------------'
              write(1,*) 'Outchannel',i,':'
              write(1,*) '-'
              write(1,*) (anglestat(i,j),j=1,6)
40         continue
           write(1,*) 'Angular distributions, cos(theta) in lab frame:'
           write(1,*) (labangle(j),j=1,6)
           write(1,*) 'Timefaults    : ',timefault
           write(1,*) 'Min/Max cm sq.: ',minsq,maxsq
           write(1,*) 'Min/Max Sigma : ',minsigma,maxsigma
           write(1,*) 'Min/Max Cosine: ',mincos,maxcos
           write(1,*) 'Gridspacing z : ',gridspz
```

The procedure checknumbers prints out statistics about particle numbers and checks them. It is also able to track particles in the grid chains, but this feature is not used here. The procedure **restat** resets the statistics and is part of the init major file.

```
           call checknumbers(0,0,0)
           call restat()
           call constant()
           outp=0
        endif
        outp=outp+1
c
10    continue
```

B.2.3 The end

Now a few final plot commands are produced:

```
c
      if (causalpl.ne.0) then
         write(4,*) 'SET LIMITS Y FROM 0 TO',(stymax+1)*gridspxy
         write(4,*) 'CONTOUR LABEL=OFF'
         write(4,*) 'CONTOUR 1 LABEL=OFF DOTDASH'
         write(4,*) 'PLOT AXES'
         write(4,*) 'TITLE X CENTER ''TIME IN FM/C'''
         write(4,*) 'TITLE Y CENTER ''SQRT(X*X+Y*Y) IN FM'''
      endif
c
      if (velocitypl.ne.0) then
         write(5,*) 'SET LIMITS Y FROM 0 TO',(veymax+1)/vgrid
         write(5,*) 'CONTOUR LABEL=OFF'
         write(5,*) 'CONTOUR 1 LABEL=OFF DOTDASH'
         write(5,*) 'PLOT AXES'
         write(5,*) 'TITLE X CENTER ''TIME IN FM/C'''
         write(5,*) 'TITLE Y CENTER ''VELOCITY IN UNITS OF C'''
      endif
```

```
c
      call date(dat)
      call time(tim)
      write(1,*) secnds(t0),': *** Program successfully terminated ***'
      write(1,*) 'Date: ',dat,' Time: ',tim
c
      stop 'That''s all folks!'
      end
```

B.2.4 The include section

The major parts of the program are included here. The following example is taken from a test version of the program:

```
* ------------------------------------------------------------------------
* Includes
* ------------------------------------------------------------------------
c Procedure to read parameters
      include 'parmin.for'
c Procedure to init particle data
      include 'init.for'
*      include 'testinit.for'
c Procedure to plot data
      include 'plotout.for'
c Procedure to move all particles according to their momenta
      include 'move.for'
c Procedure to check for scattering
c   with causality violation ...
*      include 'scatter.for'
c   ... and without
      include 'causcatter.for'
c Procedure for particle decay
      include 'decay.for'
c Function to calculate on-shell masses
      include 'mshell.for'
c Procedure that decides wether or not two particles scatter - and scatters
      include 'collide.for'
*      include 'testcollide.for'
c Procedure that completes two to two scatterings
      include 'twototwo.for'
*      include 'testtwototwo.for'
c Procedure to randomly exchange entries in permutation array
      include 'permute.for'
c Procedure to set new momenta in c.m. frame according to scattering angle
      include 'newmomenta.for'
*      include 'testnewmom.for'
c Procedure to calculate random numbers
      include 'random.for'
c Parton distribution function
      include 'distri.for'
c Procedure to check constants
      include 'constant.for'
c Procedure to remove and create particles
```

```
      include 'credel.for'
```

This is where new initialization methods have to be linked:

```
c Initialization procedures
      include 'initmeth1.for'
      include 'initmeth2.for'
      include 'initmeth3.for'
c Gauss Legendre Weights and Abscissae
      include 'gauleg.for'
c Procedures that calculate total cross sections depending on c.m. energy
c Test versions
*      include 'testsigqq.for'
*      include 'testsigqqp.for'
c Procedures doing the Monte Carlo, Testversions
*      include 'testmonqq.for'
*      include 'testmonqqp.for'
c Procedures for total cross sections and Monte Carlos by Joelle
      include 'MONTEGG.FOR'
      include 'MONTEGGQQB.FOR'
*      include 'testmonteggqqb.for'
*      include 'MONTEQG.FOR'
*      include 'testmonteqg.for'
      include 'fmonteqg.for'
      include 'MONTEQQ.FOR'
      include 'MONTEQQB.FOR'
      include 'MONTEQQBGG.FOR'
      include 'MONTEQQBP.FOR'
      include 'MONTEQQP.FOR'
      include 'TSIGGG22.FOR'
*      include 'tsiggg2205.for'
      include 'TSIGGGQQB.FOR'
      include 'TSIGQG22.FOR'
      include 'TSIGQQ22.FOR'
      include 'TSIGQQB22.FOR'
      include 'TSIGQQBGG.FOR'
      include 'TSIGQQBQPQBP.FOR'
      include 'TSIGQQP22.FOR'
```

B.3 INIT

Gerd Kortemeyer

Last revision of file: 07/21/94

In this file there are most of the routines for the initialization of the particle data except for the routine that creates the particles. This exception was made to make it possible to have more than one way of initialization.

Also the permutation arrays and other control data are initialized here.

From outside this file only the subroutine `init` is called.

B.3.1 Subroutine init

In the subroutine `init` first the number of particles for all the testruns `nq(test)` is set to zero:

```
c
c No particles so far
c
      do 10 test=1,ntr
         nq(test)=0
10    continue
```

Then the particles for the target are initialized by the subroutine `makenucleus` (see B.3.2 on page 103). All the data necessary for the initialization of this nucleus is first to be stored in the common block `create`.

This is

- The kinetic energy per nucleon of that nucleus in the lab frame.
- The initial x position of the nucleus' center in the lab frame.
- The initial z position of the nucleus' center in the lab frame. The y position is chosen to be zero always.
- The number of protons in the nucleus.
- The number of neutrons in the nucleus.
- The direction for the momentum: '1' means 'travelling in positive z direction', '-1' means 'travelling in negative z direction.
- The flag the particles from this nucleus should have. This corresponds to the array `fq` in the particle data block.

```
c
c Initialize target
c
      write(1,*) '------------------ TARGET --------------------'
c
      crelabpn=elabpntar
      crex0=0.
      crez0=0.
      crenp=nptar
      crenn=nntar
      cremodir=1.
      creflag=1
c
      call makenucleus()
```

The same procedure for the projectile:

```
c
c Initialize projectile
c
      write(1,*) '---------------- PROJECTILE -----------------'
c
```

```
      crelabpn=elabpnpro
      crex0=prox0
      crez0=proz0
      crenp=nppro
      crenn=nnpro
      cremodir=-1.
      creflag=-1
c
      call makenucleus()
```

Now the following particle data should have been initialized:

rq, pq, eq, tq, fq and nnq

Also nq should give the correct number of particles per test run.

The next thing is to initialize the other flags that are independent of the initialization mechanism:

```
c Set some flags
      call initflag()
```

These are

last, qq and pointscat

The charge-array is initialized in the subroutine initcharge:

```
c Initialize chargearray
      call initcharge()
```

As small checks charge and mass of the two nuclei are calculated from the particle data and compared to values calculated from the parameter file values:

```
c Small checks
      call initcheck()
```

The permutation arrays perm, permx, permy, permz, deltx, delty and deltz are initialized:

```
c Initialize permutations
      call initperm()
```

The cross section tables xsec... are initialized:

```
c Initialize cross section table
      call initcross()
```

The statistics with maxsigma, minsigma, timefault, ... are reset:

```
c Reset statistics
      call restat()
```

The initial gridspacings are set:

```
c
      gridspxy=2.*sqrt(maxsig/pi)
c
      write(1,*) 'Gridspacing xy           [fm]:',gridspxy
      write(1,*) '(Gridextension xy +/-',gridspxy*mgridxy,' fm)'
c
```

The gridspace in z-direction however should be at least 20 times the initial spacing between the nuclei:

```
      gridspz=2.*timstp
      if (gridspz*mgridz.le.20.*abs(proz0)) gridspz=20.*proz0/mgridz
c
      write(1,*) 'Gridspacing z            [fm]:',gridspz
      write(1,*) '(Gridextension z  +/-',gridspz*mgridz,' fm)'
c
```

For plots the z-spacing `gridz` will be used:

```
      write(1,*) '*** This gridextension will be set for all plots ***'
      gridz=gridspz
```

Finally the timestep number `ntime` is set to zero:

```
c
c Time step zero
c
      ntime=0
c
      write(1,*) secnds(t0),': Init finished'
      write(1,*)
     &    '-----------------------------------------------------------'
c
      return
      end
```

B.3.2 Subroutine makenucleus

This subroutine is calling the initialization method chosen by the user from the parameter file. It first prints the initialization values to the protocol file.

```
      write(1,*) 'Nucleus initialization, method ',initmeth,' :'
      write(1,*) '    Kin. energy per nucleon, lab. [GeV]:',crelabpn
      write(1,*) '    Position x in lab frame       [fm] :',crex0
      write(1,*) '    Position z in lab frame       [fm] :',crez0
      write(1,*) '    Number of protons                  :',crenp
      write(1,*) '    Number of neutrons                 :',crenn
      write(1,*) '    Momentum direction (forward/backw.):',cremodir
      write(1,*) '    Parton flag                        :',creflag
```

Now the call of the chosen initialization method is performed. This list has to be expanded as soon as a new initialization method is to be implemented:

```
c
      if (initmeth.eq.1) call initmeth1()
c
```

B.3.3 Subroutine initflag

In this subroutine the last scattering partner is set to 'itself', and the last scattering point is set to 'nowhere':

```
      write(1,*) secnds(t0),': Setting flags and charges'
c For all test runs
      do 10 test=1,ntr
c
c For all existing particles in test run
c
         do 20 i=1,nq(test)
c Last scattering with itself
            last(test,i)=i
c Not scattered before
            pointscat(test,i)=-1
20       continue
```

Also, for unused entries of the particle arrays the entry `last` is set to '-1'. This is done for creating and deleting particles, see B.12 on page 149.

```
c
c Set remaining particle flags to 'not existing'
c These entries are flaged 'last=-1'
c
         do 30 i=nq(test)+1,mquark
            last(test,i)=-1
30       continue
c
c Number of particles
c
         lastnumber(test)=nq(test)
c
10    continue
c
      return
      end
```

B.3.4 Subroutine initcharge

In this subroutine the array charge is initialized:

```
      charge(0 )= 0.
      charge(1 )= 2./3.
      charge(-1)=-2./3.
      charge(2 )=-1./3.
      charge(-2)= 1./3.
      charge(3 )= 2./3.
      charge(-3)=-2./3.
      charge(4 )=-1./3.
      charge(-4)= 1./3.
      charge(5 )= 2./3.
      charge(-5)=-2./3.
      charge(6 )=-1./3.
      charge(-6)= 1./3.
```

B.3.5 Subroutine initcheck

This subroutine calculates the total charge and the total mass of all particles and compares it to the values being calculated from the data in the parameter file:

```
      write(1,*) secnds(t0),': Checking charge and mass'
      do 10 test=1,ntr
         qchar=0.
         mass=0.
         do 200 i=1,nq(test)
            qchar=qchar+charge(tq(test,i))
            mass=mass+sqrt(eq(test,i)**2
     &           -pq(test,i,1)**2-pq(test,i,2)**2-pq(test,i,3)**2)
200      continue
c
         write(1,*) 'Check charge :',qchar,' (',nptar+nppro,')'
         write(1,*) 'Check mass   :',mass,' (',
     &                 (nptar+nppro+nntar+nnpro)*manuc,')'
10    continue
```

B.3.6 Subroutine initperm

This subroutine initializes the permutation arrays by setting each entry to its own index number.

```
      do 400 i=1,mquark
         perm(i)=i
400   continue
      do 410 i=-mgridxy,mgridxy
         permx(i)=i
         permy(i)=i
410   continue
      do 420 i=-mgridz,mgridz
         permz(i)=i
420   continue
      do 430 i=1,3
         deltx(i)=2-i
         delty(i)=2-i
         deltz(i)=2-i
430   continue
```

B.3.7 Subroutine initcross

In this subroutine the cross section arrays `xsec` and the maximum cross section `maxsig` are initialized. The cross sections are calculated on a mesh with meshpoints `eps+sqstep*(i-1)**4`, where `i` is the index of the mesh point. All together the mesh has `maxpoints` points.

```
      integer i,j,k
c
      write(1,*) secnds(t0),': Initializing cross sections'
c
      write(1,*) 'Max. energy [GeV**2]:',
     &             sqstep*(maxpoints-1.)**4
c
c Maximum total cross section to be found:
c
      maxsig=0.
```

The cross section arrays are first set to the corresponding partial cross sections. The variables `tin1`, `tin2` and `tout1` are used by the cross section routines to determine the on-shell masses of the partons involved.

The big loop runs over all mesh point indices:

```
c
c First calculate cross sections
c
      do 20 i=1,maxpoints
```

The energy corresponding to the index is calculated using the above formula:

```
         tpsq=eps+(i-1.)**4*sqstep
         call tsiggg22()
         xsecgg22(i)=tpsigma
```

The next loop runs over all possible incoming quark types:

```
         do 20 tin1=1,maxtype
            call tsigqq22()
            xsecqq22(tin1,i)=tpsigma
            if (tpsigma.gt.maxsig) maxsig=tpsigma
            call tsigqg22()
            xsecqg22(tin1,i)=tpsigma
            if (tpsigma.gt.maxsig) maxsig=tpsigma
            call tsigqqb22()
            xsecqqb22(tin1,i)=tpsigma
            call tsigqqbgg()
            xsecqqbgg(tin1,i)=tpsigma
```

For `tsigggqqb` the varible `tout1` has to be set according to the types of the outgoing partons:

```
            tout1=tin1
            call tsigggqqb()
            xsecggqqb(tout1,i)=tpsigma
```

In the following cross sections two different quark types are involved. For tsigqqp22 those two different quarks are in the incoming channel, so also tin2 has to be set, for tsigqqbqpqbp one quark type has to be set for the incoming channel and one for the outgoing channel.

```
          do 20 j=1,maxtype
             if (tin1.ne.j) then
                tin2=j
                call tsigqqp22()
                xsecqqp22(tin1,tin2,i)=tpsigma
                tout1=j
                call tsigqqbqpqbp()
                xsecqqbqpqbp(tin1,tout1,i)=tpsigma
             else
                xsecqqp22(tin1,j,i)=0.
                xsecqqbqpqbp(tin1,j,i)=0.
             endif
20    continue
```

The partial cross sections at the minimum and maximum energy of the mesh are written to the protocol file:

```
c
c Control output
c
      write(1,*) '-'
      write(1,*) 'Parton label (6, 5, 4, 3, 2, 1, 0, -1, ......, -6)'
      write(1,*) '         for (t, b, c, s, d, u, g, u_bar, ..., t_bar)'
      write(1,*) '-'
      write(1,*) 'Elastic channels at min/max energies [fm**2]:'
      write(1,*) '--------------------------------------------'
      write(1,*) 'gg->gg',xsecgg22(1),xsecgg22(maxpoints)
      write(1,*) 'qq->qq'
      do 30 i=1,maxtype
         write(1,*) i,xsecqq22(i,1),xsecqq22(i,maxpoints)
30    continue
      write(1,*) 'qg->qg'
      do 40 i=1,maxtype
         write(1,*) i,xsecqg22(i,1),xsecqg22(i,maxpoints)
40    continue
      write(1,*) 'qqb->qqb'
      do 50 i=1,maxtype
         write(1,*) i,xsecqqb22(i,1),xsecqqb22(i,maxpoints)
50    continue
      write(1,*) 'qqp->qqp'
      do 60 i=1,maxtype
         do 60 j=1,maxtype
            write(1,*) i,j,xsecqqp22(i,j,1),xsecqqp22(i,j,maxpoints)
60    continue
      write(1,*) 'Inelastic channels at min/max energies [fm**2]:'
      write(1,*) '--------------------------------------------'
      write(1,*) 'qqb->gg'
      do 70 i=1,maxtype
         write(1,*) i,xsecqqbgg(i,1),xsecqqbgg(i,maxpoints)
70    continue
```

```
      write(1,*) 'gg->qqb'
      do 80 i=1,maxtype
         write(1,*) i,xsecggqqb(i,1),xsecggqqb(i,maxpoints)
80    continue
      write(1,*) 'qqb->qpqbp'
      do 90 i=1,maxtype
         do 90 j=1,maxtype
            write(1,*) i,j,xsecqqbqpqbp(i,j,1),
     &                     xsecqqbqpqbp(i,j,maxpoints)
90    continue
```

Now for a given combination of incoming partons the partial cross sections for all combinations of outgoing partons are added up in a certain sequence. Finally the entry for a certain partial cross section is the cross section in question plus the sum of all preceeding cross sections in that sequence.

For the incoming channel gg the open output channels are gg or $q\bar{q}$. The partial cross sections for two incoming gluons are added up in the order $gg \to gg$ (xsecgg22), $gg \to u\bar{u}$, $gg \to d\bar{d}$, ..., $gg \to t\bar{t}$ (xsecggqqb).

For example in xsecggqqb(3,energy) one will find the partial cross section for $gg \to s\bar{s}$ plus the partial cross sections for $gg \to d\bar{d}$, $gg \to u\bar{u}$ and $gg \to gg$.

For the incoming channels qg and qq only the elastic channel is open.

For the incoming channel $q\bar{q}$ the partial cross sections are added up in the following order: $q\bar{q} \to q\bar{q}$ (xsecqqb22), $q\bar{q} \to gg$ (xsecqqbgg), $q\bar{q} \to q'\bar{q}'$ (xsecqqbqpqbp). The cross section array entries for $q\bar{q} \to q'\bar{q}'$ are added up in the order $q\bar{q} \to u\bar{u}$, $q\bar{q} \to d\bar{d}$ and set to zero when $q = q'$.

As a final example in xsecqqbqpqbp(2,4,energy) one will find the partial cross section for $d\bar{d} \to b\bar{b}$ plus the following partial cross sections: $d\bar{d} \to s\bar{s}$ and $d\bar{d} \to u\bar{u}$ from xsecqqbqpqbp, $d\bar{d} \to gg$ from xsecqqbgg and $d\bar{d} \to d\bar{d}$ from xsecqqb22.

As one can see, the partial cross sections always add up to the total cross section in the last entry of that sequence.

```
c
c Now pile them up
c
c Elastic channels first, then inelastic
c
      do 100 i=1,maxpoints
         xsecggqqb(1,i)=xsecggqqb(1,i)+xsecgg22(i)
         do 110 j=2,maxtype
            xsecggqqb(j,i)=xsecggqqb(j,i)+xsecggqqb(j-1,i)
            if (xsecggqqb(j,i).gt.maxsig) maxsig=xsecggqqb(j,i)
110      continue
         do 120 j=1,maxtype
            xsecqqbgg(j,i)=xsecqqbgg(j,i)+xsecqqb22(j,i)
            xsecqqbqpqbp(j,1,i)=xsecqqbqpqbp(j,1,i)+xsecqqbgg(j,i)
            do 120 k=2,maxtype
               xsecqqbqpqbp(j,k,i)=xsecqqbqpqbp(j,k,i)+
     &                             xsecqqbqpqbp(j,k-1,i)
               if (xsecqqbqpqbp(j,k,i).gt.maxsig)
     &            maxsig=xsecqqbqpqbp(j,k,i)
120      continue
100   continue
```

The total cross sections, again at minimum and maximum mesh energy, are given out to the protocol file:

```
c
c Control output
c
      write(1,*) 'Total cross section at min/max energies [fm**2]:'
      write(1,*) '-----------------------------------------------'
      write(1,*) 'gg->anything',
     &           xsecggqqb(maxtype,1),xsecggqqb(maxtype,maxpoints)
      write(1,*) 'qqb->anything'
      do 200 i=1,maxtype
         write(1,*) i,xsecqqbqpqbp(i,maxtype,1),
     &              xsecqqbqpqbp(i,maxtype,maxpoints)
c
200   continue
c
      write(1,*) 'Maximum total cross section found [fm**2]:',maxsig
c
      return
      end
```

B.3.8 Subroutine rancor

This subroutine randomly chooses x, y and z such that

$$x^2 + y^2 + z^2 \leq 1 \,,$$

$$0 \leq x,\ y,\ z \leq 1 \,.$$

```
      subroutine rancor(x,y,z)
c
      double precision x,y,z
c
10    x=1.-2.*ran1(iseed)
      y=1.-2.*ran1(iseed)
      z=1.-2.*ran1(iseed)
      if (x*x+y*y+z*z.gt.1.) goto 10
c
      return
      end
```

B.3.9 Subroutine restat

This subroutine resets the statistics values being written to the protocol file whenever there is a 'big' output.

```
c
c Reset min/max
c
      maxsigma=0.
      minsigma=maxsig
      maxsq=0.
      minsq=1000000000.
      mincos=1.
      maxcos=-1.
c
c Reset multistage and credel statistics
c
      do 300 i=1,ntr
         nsta0(i)=0
         nsta1(i)=0
         nsta2(i)=0
         nsta3(i)=0
         nsta4(i)=0
         crenumber(i)=0
         delnumber(i)=0
300   continue
```

The scattering channel statistics **chscat** are reset. In this array scattering events in the different channel combinations will be counted:

```
      do 310 i=1,maxchannel
         do 310 j=1,maxchannel
            chscat(i,j)=0
310   continue
```

The angular distribution bins are emptied:

```
      do 410 i=1,maxchannel
         do 410 j=1,6
            anglestat(i,j)=0
410   continue
c
      do 420 j=1,6
         labangle(j)=0
420   continue
```

The impact parameter statistics are reset:

```
      do 500 i=0,20
         impact(i)=0
500   continue
```

Finally, the number of timefaults is set to zero:

```
      timefault=0
```

B.4 INITMETH1

Gerd Kortemeyer

Last revision of the file: 01/20/94

This subroutine is an example for what requirements an initialization routine for nuclei must meet. It represents a very simple version that only initializes the valenz quarks and can be used as a primer when a more sophisticated initialization routine is to be written.

In general, an initialization routine must set the following variables:

`rq`, `pq`, `eq`, `tq`, `fq` and `nnq`

Also `nq` must give the correct number of particles per test run.

B.4.1 Local variables

In addition to `stdec.for` the following variables are defined:

```
c Radius of nucleus in xy and z
      double precision radxy, radz
c Radius of nucleon in z
      double precision radnucz
c Momentum per nucleon
      double precision monuc
c Total energy of nucleon in lab frame
      double precision etlabn
c Momentum per quark
      double precision moqua
c Energy per quark
      double precision eqqua
c Beta and gamma for transformation from lab to rest
      double precision beta, gamma
c Test run
      integer test
c Nucleon counter
      integer ncount
c do variables
      integer i,j
c Coordinates
      double precision x,y,z,xc,yc,zc
```

B.4.2 Initialization of the particles

First, it is important that the initialization routine clearly writes to the protocol file which kind of initialization is chosen:

```
c
      write(1,*) '>>> -----------------------------------'
      write(1,*) '>>> Initializing Valenz-Quarks only ...'
      write(1,*) '>>> -----------------------------------'
c
```

Now a few kinematical calculations are performed:

```
c E=Ekin+m^2
      etlabn=crelabpn+manuc
      write(1,*) 'Total energy per nucleon, lab frame [GeV]:',etlabn
c E^2-p^2=m^2
c p^2=E^2-m^2
      monuc=cremodir*sqrt(etlabn*etlabn-manuc*manuc)
      write(1,*) 'Total momentum per nucleon, lab   [GeV/c]:',monuc
c beta=p/E
      beta=monuc/etlabn
      write(1,*) 'Beta for transformation from rest frame  :',beta
c gamma=1/sqrt(1-beta^2)
      gamma=1./sqrt(1.-beta*beta)
c r=r0*A^(1/3)
      radxy=r0*(crenp+crenn)**(1./3.)
      write(1,*) 'Radius of nucleus xy                 [fm]:',radxy
c rz=r/gamma
      radz=radxy/gamma
      write(1,*) 'Radius of nucleus z (contracted)     [fm]:',radz
c
      write(1,*) 'Radius of nucleon xy                 [fm]:',initrnuc
      radnucz=initrnuc/gamma
      write(1,*) 'Radius of nucleon z (contracted)     [fm]:',radnucz
      write(1,*) '---'
c
      eqqua=etlabn/3.
      write(1,*) 'Energy per quark                    [GeV]:',eqqua
c
      moqua=monuc/3.
      write(1,*) 'Momentum per quark                [GeV/c]:',moqua
```

B.4.3 Creating particles

Now for all testruns the particles are created:

```
c
c For all testruns
c
      do 10 test=1,ntr
         write(1,*) '----- Testrun : ',test
c Reset nucleon counter
         ncount=0
```

From here on the initialization methods might differ. First, all protons are created, their constituents are just being thrown into one and the same arrays – only the nucleon number `nnq` still identifies which nucleon they belong to, while the flag `fq` identifies which of the two nuclei, target or projectile, their nucleon belongs to:

```
c
c Do the protons
c --------------
         do 20 i=1,crenp
            ncount=ncount+1
```

```
c Where to place nucleon?
            call rancor(x,y,z)
            xc=crex0+x*(radxy-initrnuc)
            yc=      y*(radxy-initrnuc)
            zc=crez0+z*(radz -radnucz)
c xc, yc and zc are the center coordinates of the nucleon
c Now put 3 valenz quarks into nucleon
            do 30 j=1,3
c One particle more
               nq(test)=nq(test)+1
c Position
               call rancor(x,y,z)
               rq(test,nq(test),1)=xc+x*initrnuc
               rq(test,nq(test),2)=yc+y*initrnuc
               rq(test,nq(test),3)=zc+z*radnucz
c Momentum
               pq(test,nq(test),1)=0.
               pq(test,nq(test),2)=0.
               pq(test,nq(test),3)=moqua
c Energy
               eq(test,nq(test))=eqqua
c Flag
               fq(test,nq(test))=creflag
c Nucleon number
               nnq(test,nq(test))=ncount
30          continue
c Now make them up and down
            tq(test,nq(test)-2)=1
            tq(test,nq(test)-1)=2
            tq(test,nq(test)  )=1
20       continue
```

The creation of the neutrons of course is similar except for the valenz quark types:

```
c
c Do the neutrons
c ---------------
         do 40 i=1,crenn
            ncount=ncount+1
c Where to place nucleon?
            call rancor(x,y,z)
            xc=crex0+x*(radxy-initrnuc)
            yc=      y*(radxy-initrnuc)
            zc=crez0+z*(radz -radnucz)
c xc, yc and zc are the center coordinates of the nucleon
c Now put 3 valenz quarks into nucleon
            do 50 j=1,3
c One particle more
               nq(test)=nq(test)+1
c Position
               call rancor(x,y,z)
               rq(test,nq(test),1)=xc+x*initrnuc
               rq(test,nq(test),2)=yc+y*initrnuc
               rq(test,nq(test),3)=zc+z*radnucz
```

```
c Momentum
               pq(test,nq(test),1)=0.
               pq(test,nq(test),2)=0.
               pq(test,nq(test),3)=moqua
c Energy
               eq(test,nq(test))=eqqua
c Flag
               fq(test,nq(test))=creflag
c Nucleon number
               nnq(test,nq(test))=ncount
50          continue
c Now make them up and down
            tq(test,nq(test)-2)=2
            tq(test,nq(test)-1)=1
            tq(test,nq(test)  )=2
40        continue
c
10      continue
```

B.4.4 The end

Finally as a small check for the last testrun the number of created nucleons is compared to the expected value:

```
c
      write(1,*) ncount,'(',crenp+crenn,') nucleons.'
c
      return
      end
```

B.5 MOVE

Gerd Kortemeyer

Last revision of the file: 04/04/93

This subroutine moves all particles according to their momenta and also initializes the pointer structure for the assignment of the particles to the boxes of the spatial grid.

B.5.1 Local variables

```
c Do variables
      integer i,j,k,m,nx,ny,nz
c Permutation subset with numbers only from 1 to nq(test_run_number)
      integer subperm(mquark)
c Array with current endparticles of gridboxes
      integer endpart
     & (-mgridxy:mgridxy,-mgridxy:mgridxy,-mgridz:mgridz)
c Velocity 3 vector
      real v(3)
c Counter for particles having left boxes
```

```
      integer outb
c Maximum angle, angle
      double precision mtheta,theta
```

B.5.2 Determining the new gridsize

In order to adjust the gridsize during runtime the greatest angle between the direction any particle is moving in and the z-direction has to be determined:

```
c
c Determine new gridspacing
c
      mtheta=0.
c
      do 100 i=1,ntr
         do 100 j=1,nq(i)
            if (last(i,j).ne.-1) then
c Get angle
               theta=abs(
     &      atan(sqrt(pq(i,j,1)*pq(i,j,1)+pq(i,j,2)*pq(i,j,2))/
     &                        pq(i,j,3)))
               if (theta.gt.mtheta) mtheta=theta
            endif
100   continue
```

With **theta**= $\theta_{\max}$ being the maximum angle any particle has from the initial direction, the grid extension in z direction is adjusted to $2t_{\text{step}} \cdot c + 2\sqrt{\frac{\sigma_{\max}}{\pi}}\sin(\theta_{\max})$:

```
      gridspz=2.*(timstp+sqrt(maxsig/pi)*sin(mtheta))
```

B.5.3 Moving the particles

First the variable outb is set to zero here. It will count the particles that have left the grid area.

```
c
      outb=0
c
```

Over all there is a loop over all test runs:

```
c Loop for all active test-runs:
      do 10 i=1,ntr
```

For the respective test run the gridroots are set to 'empty box'.

```
c Reset gridroots and endpart
         do 20 j=-mgridxy,mgridxy
            do 20 k=-mgridxy,mgridxy
               do 20 m=-mgridz,mgridz
                  gridroot(i,j,k,m)=-1
20       continue
```

In the integer array subperm the permuted numbers from 1 to nq(i) are initialized in the entries 1 to nq(i). i is the number of the test run, nq(i) is the number of particles in this run. This is a 'subpermutation' of the permutation perm that has the numbers 1 to mquark in the entries 1 to mquark. subperm is designed for the random access of the nq(i) particles in the actual test run.

```
c Initialize permutation subset
         j=1
         do 30 k=1,mquark
            if (perm(k).le.nq(i)) then
               subperm(j)=perm(k)
               j=j+1
            endif
30       continue
```

If the subpermutation was initialized correctly, the value of j is nq(i)+1:

```
c Check subperm
         if (j.ne.nq(i)+1) then
            write(1,*) 'Sub-Permutation wrong, j=',j
            write(1,*) '                  Test-Run:',i
            write(1,*) '      Number of particles:',nq(i)
            stop
         endif
```

Now there is a loop over all particles in the test run, but the actual index j for the particle that is to be moved is taken from the subpermutation. This is done in order to avoid that the chain of pointers for the particles in the boxes has the same order in subsequent timesteps:

```
c Loop for all particles in that run:
         do 10 m=1,nq(i)
c Particle numbers chosen from permutation array
            j=subperm(m)
```

It then has to be checked if the particle is actually existing:

```
c Particle present?
            if (last(i,j).ne.-1) then
```

See A.7 on page 84 and B.1.3 on page 87 for details.

The moving of the particle is done using

$$\vec{r}(t+\Delta t) \approx \vec{r}(t) + \vec{v}(t)\cdot\Delta t = \vec{r}(t) + \frac{\vec{p}(t)}{E(t)}\cdot\Delta t \; .$$

```
c Loop for all components of the 3-vector
               do 40 k=1,3
c
                  rq(i,j,k)=rq(i,j,k)+pq(i,j,k)*timstp/eq(i,j)
c
40             continue
```

B.5.4 Assigning the particles to the boxes

The indices of the box the particle is in now are calculated as the coordinate divided by the respective gridspacing.

```
c In which box is the particle now?
c
               nx=nint(rq(i,j,1)/gridspxy)
c
               ny=nint(rq(i,j,2)/gridspxy)
c
               nz=nint(rq(i,j,3)/gridspz)
```

Now it is to be checked if the particles left the grid area. If so, they are put into the outmost box and the counter `outb` is incremented.

```
c Did particles leave the box?
               if (nx.gt.mgridxy) then
                  nx=mgridxy
                  outb=outb+1
               endif
               if (nx.lt.-mgridxy) then
                  nx=-mgridxy
                  outb=outb+1
               endif
               if (ny.gt.mgridxy) then
                  ny=mgridxy
                  outb=outb+1
               endif
               if (ny.lt.-mgridxy) then
                  ny=-mgridxy
                  outb=outb+1
               endif
               if (nz.gt.mgridz) then
                  nz=mgridz
                  outb=outb+1
               endif
               if (nz.lt.-mgridz) then
                  nz=-mgridz
                  outb=outb+1
               endif
```

Now the particle certainly has a valid box index it is time to incorporate it into the pointer structure.

The pointer array `endpart` is used to always point to the last particle being found in the respective box during initialization of the pointer structure, it is not needed once the initialization is finished.

If the actual particle is the first particle found in its box both `gridroot` and `endpart` will be initialized pointing to that particle.

If the box wasn't empty the previously found particles pointer is set to the actual particle.

```
c Is it the first particle in that box?
               if (gridroot(i,nx,ny,nz).lt.0) then
                  gridroot(i,nx,ny,nz)=j
```

```
                endpart(nx,ny,nz)=j
             else
                nextgrid(i,endpart(nx,ny,nz))=j
             endif
```

Finally, the actual particles pointer is set to 'nowhere' since it is not sure if there will be another particle found. If there is one more particle in the box, the actual particles pointer will be set to it at the time the next particle is found by the above statement `nextgrid(i,endpart ....`

Also the `endpart` pointer is set to actual particle number.

```
c Set nextgrid to nowhere, endpart to j
             nextgrid(i,j)=-1
             endpart(nx,ny,nz)=j
```

B.5.5 The end

Now the `if` is closed and the label for the two loops over the test run and the particles is set:

```
           endif
10    continue
```

Also a warning is given out if one or more particles left the grid:

```
c Write warnings
      if (outb.ne.0) write(1,*)
     & '!!!',outb,' grid boundary violations in timestep',ntime
c
```

Bye bye...

```
      return
      end
```

B.6 SCATTER

Gerd Kortemeyer

Last revision of file: 09/19/94

This subroutine is responsible for finding all particle combinations that are able to scatter. Candidates are delievered to subroutine `collide` where the physics tests for scattering are run and in case the scattering is executed.

B.6.1 Local variables

Besides `stdec.for` the following local variable definitions are made:

```
c do variables
      integer i,j,k,l,m,n
c box numbers of center box
      integer ni,nj,nk
c box numbers of second box
      integer mi,mj,mk
c bases for linear index
      integer base,basesq
c linear index of first and second box
      integer linfirst,linsecond
c Data about first particle
      integer lastp1,flagp1
```

B.6.2 Initialization of variables

First the subroutine defines the variables

```
      base=2*mgridxy+1
      basesq=base*base
```

that will be used later to calculate an identification number for the boxes. The compiler will treat both variables like constants.

B.6.3 Choosing the boxes

For choosing the boxes first a big loop over all test runs is executed. The loop variable is chosen to be `tprn` so automatically the right test run number can be found in the common `twopart`.

```
c
c For all test runs active
c
      do 10 tprn=1,ntr
```

The array `alr`, that is used to flag particles that already have scattered in this timestep, is set to zero, meaning 'not scattered'

```
c No scattering so far in this run
         do 20 j=1,nq(tprn)
            alr(tprn,j)=0
20       continue
```

Next, all possible center boxes are looked at. The mapping from the do-variables `i,j,k` to the box indizes `ni,nj,nk` is done by means of the permutation arrays `permx,permy,permz`:

```
c Choose center box, not including outmost boxes
         do 30 i=-mgridxy+1,mgridxy-1
            do 30 j=-mgridxy+1,mgridxy-1
               do 30 k=-mgridz+1,mgridz-1
                  ni=permx(i)
                  nj=permy(j)
                  nk=permz(k)
```

The outmost boxes are excluded from the loop. The mapping through permx, permy and permz generates no problems here since it is taken care in subroutine permute that the indices for the outmost boxes are not permuted.

Now gridroot(tprn,ni,nj,nk) points to the first particle being found in that box. However, if the box is empty the gridroot value is '-1'. In this case it makes no sense to use this box as the center box and the program jumps to label 40 that immediately preceeds the loop label 30: The loop chooses the next center box.

```
c Center box empty?
                  if (gridroot(tprn,ni,nj,nk).lt.0) goto 40
```

Without further effort every box combination would be checked twice: One time when it is the center box, the other time when it is the neighbor box of another center box. If the boxes had one index instead of three the problem would be easy to solve: One could simply only use combinations where the index of the first box is smaller than or equal to the index of the second box.

Nevertheless a single linear index can be produced by the following mapping:

```
c Calculating linear index of first box
                  linfirst=(nk+mgridz)*basesq
     &                    +(nj+mgridxy)*base
     &                    + ni+mgridxy
```

2·mgridxy+1 is chosen as base, basesq is base^2. The adding of mgridz and mgridxy is done to make for example nj+mgridxy range from 0 to 2·mgridxy.

Now a loop for all neighboring boxes is set up. All together there are 27 box combinations to be checked. In order to do this in a random sequence the mapping via deltax,deltay,deltaz is used, the indizes for the second box are mi,mj,mk afterwards:

```
c Second box
         do 50 l=1,3
            do 50 m=1,3
               do 50 n=1,3
                  mi=ni+deltx(l)
                  mj=nj+delty(m)
                  mk=nk+deltz(n)
```

If the second box is empty the program jumps to label 60 that immediately preceeds the loop label 50 for the second box.

```
c Second box empty?
                  if (gridroot(tprn,mi,mj,mk).lt.0) goto 60
```

For this box also a linear index is calculated, afterwards the indizes are compared. If the first index is bigger than the second index it is jumped to label 60 for the next second box:

```
c Calculating linear index of second box
                  linsecond=(mk+mgridz)*basesq
     &                    +(mj+mgridxy)*base
     &                    + mi+mgridxy
c Now we have two non-empty boxes with indices ni,nj,nk and mi,mj,mk.
c As you don't want to check box combinations twice, check only the
c upper diagonal.
                  if (linfirst.gt.linsecond) goto 60
```

B.6.4 Combination of all particles from the boxes

As already explained the particles from the boxes are accessed by following a chain of pointers starting with the `gridroot`. For the actual particle number directly the variables `tpp1` and `tpp2` from the common block `twopart` are used. The label 80 is used for a handmade loop that runs over all `tpp1`, the label 70 is used for a loop over all `tpp2`.

The labels 90 for the first particle and 100 for the second particle are used to skip the respective particle and go on with the next one.

```
c Combine all particles from one box with the other boxes particles.
                  tpp1=gridroot(tprn,ni,nj,nk)
80                continue
```

Now the formal reasons against scattering explained in A.3 on page 81 have to be checked. As already explained in the array `alr` for particles not scattered in this timestep one can find '0', as soon as they have scattered '1'.

```
c Particle 1 has already scattered - next one!
                     if (alr(tprn,tpp1).ne.0) goto 90
c
                     lastp1=last(tprn,tpp1)
                     flagp1=fq(tprn,tpp1)
c
c Start loop for second particle
                     tpp2=gridroot(tprn,mi,mj,mk)
c Particle loop, tpp2
70                   continue
c Particle 2 already scattered? Next particle!
                        if (alr(tprn,tpp2).ne.0) goto 100
```

In the array `fq` for all particles that never have scattered one can find '-1' if they originally belonged to the projectile, and '1' if they originally belonged to the target. Scattered particles have `fq`=0.

```
c Same nucleus?
                        if (flagp1*fq(tprn,tpp2).eq.1) goto 100
```

In the array `last` the number of the last scattering partner of the particles can be found.

```
c Particle 2 scattered with 1 recently?
                        if (tpp2.eq.lastp1) goto 100
c Particle 1 scattered with 2 recently?
                        if (tpp1.eq.last(tprn,tpp2)) goto 100
c
```

If all this requirements are met the subroutine `collide` is called.

```
c
c ----------------------------------------------------------------------
c Collide them!
                           call collide()
                           if (alr(tprn,tpp1).ne.0) goto 90
c
c ------------------------------------------------------------------------
c
```

After the particle combination is checked, the next particles are found by the latter particles' `nextgrid` pointers. It is important that the procedures for creating and deleting particles which might be called within `collide` do not change the `nextgrid`-pointers of the chosen particles, see B.12 on page 149. If the `nextgrid`-pointers have the value '-1' no more particles are in the respective box:

```
100                        tpp2=nextgrid(tprn,tpp2)
                        if (tpp2.ge.0) goto 70
c
90                      tpp1=nextgrid(tprn,tpp1)
                     if (tpp1.ge.0) goto 80
c
c Second box
60                   continue
50         continue
c ------------->
c Center box
40                   continue
30         continue
c Test-Runs
10    continue
```

Goodbye:

```
      return
      end
```

B.7 CAUSCATTER

Gerd Kortemeyer

Last revision of file: 05/04/94

This subroutine is responsible for finding all particle combinations that are able to scatter. Candidates are delievered to subroutine `collide` where the physics tests for scattering are run and in case the scattering is executed.

It is a blueprint of the subroutine `scatter`, see B.6 on page 118 with the exception, that collisions that would lead to signal velocities faster than the speed of light are forbidden.

This file can be linked instead of `scatter` from the main file.

The differences are:

B.7.1 Local variables

```
...
c Scattering midpoint, velocity
      double precision midp(3),velo
...
```

B.7.2 Combination of all particles from the boxes

As the last check before the collision routine is called, the signal speed is calculated for both partons involved. If one of the signals would be too fast, the collision is forbidden.

The collision is assumed to take place in the middle between the partons:

```
...
c Okay, midpoint calculation
                       midp(1)=(rq(tprn,tpp1,1)+rq(tprn,tpp2,1))/2.
                       midp(2)=(rq(tprn,tpp1,2)+rq(tprn,tpp2,2))/2.
                       midp(3)=(rq(tprn,tpp1,3)+rq(tprn,tpp2,3))/2.
c
```

For the actual calculation the parton data entries `lastpoint` and `lasttime` are used. In them the midpoint coordinates of the most recent collision the parton was involved in are stored.

```
c Faster than the speed of light ... ???
c
c < ----
      if (nnq(tprn,tpp1).eq.0) then
         velo=sqrt( (midp(1)-lastpoint(tprn,tpp1,1))**2
     &             +(midp(2)-lastpoint(tprn,tpp1,2))**2
     &             +(midp(3)-lastpoint(tprn,tpp1,3))**2 ) /
     &            ( (ntime  - lasttime(tprn,tpp1))*timstp )
         if (velo.gt.1.) goto 100
      endif
c
      if (nnq(tprn,tpp2).eq.0) then
         velo=sqrt( (midp(1)-lastpoint(tprn,tpp2,1))**2
     &             +(midp(2)-lastpoint(tprn,tpp2,2))**2
     &             +(midp(3)-lastpoint(tprn,tpp2,3))**2 ) /
     &            ( (ntime  - lasttime(tprn,tpp2))*timstp )
         if (velo.gt.1.) goto 100
      endif
c ---- >
c
c ------------------------------------------------------------------------
c Collide them!
...
```

B.8 COLLIDE

Gerd Kortemeyer

Last revision of file: 08/17/94

This subroutine decides wether or not two particles, that have been determined by the subroutine `scatter`, will collide. It does this by performing a multistage check, the subroutine is left immediately if one of the requirements is not met. If the two particles scatter the output channel is chosen and a corresponding other routine called.

B.8.1 Subroutine collide

Local variables

The file `stdec.for` is included.

```
c Scalar product rp in cm-frame
      double precision rpcm1,rpcm2
c Distance vector cm-frame one timestep later
      double precision dvcm(3)
c Transformed time step
      double precision cmtimstp
c Factors in transformation formula
      double precision scalbetax,a
```

Checks in the lab frame

To check the multistage statistics everytime the routine is entered `nsta0` is increased:

```
c
c Statistics: entered routine
c
      nsta0(tprn)=nsta0(tprn)+1
```

The first checks take place in the lab (or 'computer') frame. The distance vector `tprl` is calculated as the difference of the two particles coordinate vectors. `tpdq` is the square of the difference vector.

```
c
c Calculate distance vector lab and distance lab
c ----------------------------------------------
c
      tprl(1)=rq(tprn,tpp1,1)-rq(tprn,tpp2,1)
      tprl(2)=rq(tprn,tpp1,2)-rq(tprn,tpp2,2)
      tprl(3)=rq(tprn,tpp1,3)-rq(tprn,tpp2,3)
c
      tpdq=tprl(1)*tprl(1)+tprl(2)*tprl(2)+tprl(3)*tprl(3)
```

Statistics for minimum and maximum distance in the lab frame are being set:

```
c
      if (tpdq.le.mindq) mindq=tpdq
      if (tpdq.ge.maxdq) maxdq=tpdq
c
```

It is of great importance that the spatial distance in the c.m.-frame is always bigger than distance in lab-frame! This is because the time component of the distance is zero in the lab frame.

The spatial distance in the c.m.-frame is also always bigger than or equal to the impact parameter, because the impact parameter vector is only a projection of the distance vector in the c.m.-frame. But in case of a possible scattering event the point of closest approach has to be reached as another condition, so if scattering is possible, the c.m.-distance equals the impact parameter and finally the distance in the lab frame will be smaller than the impact parameter.

If however the distance in the lab frame is bigger than the maximum impact parameter given by the maximum cross section maxsig, that is determined during the initialization of the cross section table, no scattering is possible:

```
c
c ========================================================================
c 1st check                                                             !!
c ---------                                                             !!
c Distance in c.m.-frame is always bigger than distance in lab-frame!   !!
c Distance in c.m.-frame always bigger or equal impact parameter.       !!
c Scattering only when closest approach in c.m.-system ==> scatter      !!
c only when c.m.-distance equals impact parameter.                      !!
c                                                                       !!
      if (tpdq.gt.maxsig/pi) return                                     !!
```

If the first test was passed, the statistics for this test are incremented:

```
c Increase statistics: Passed first test in this test run               !!
      nsta1(tprn)=nsta1(tprn)+1                                         !!
c ========================================================================
```

Finding the boost velocity

The transformation between c.m.- and lab frame are given by the following equations:

$$E_{\text{cm}} = \gamma(E_{\text{lab}} - \vec{\beta}\vec{p}_{\text{lab}}) \tag{B.1}$$

$$\vec{p}_{\text{cm}} = \vec{p}_{\text{lab}} + \frac{\gamma-1}{\beta^2}(\vec{\beta}\vec{p}_{\text{lab}})\vec{\beta} - \gamma\vec{\beta}E \tag{B.2}$$

$$E_{\text{lab}} = \gamma(E_{\text{cm}} + \vec{\beta}\vec{p}_{\text{cm}}) \tag{B.3}$$

$$\vec{p}_{\text{lab}} = \vec{p}_{\text{cm}} + \frac{\gamma-1}{\beta^2}(\vec{\beta}\vec{p}_{\text{cm}})\vec{\beta} + \gamma\vec{\beta}E_{\text{cm}} \tag{B.4}$$

The only question is what $\vec{\beta}$ for the intended transformation is. The c.m.-frame is defined by

$$\vec{p}_{1\text{cm}} = -\vec{p}_{2\text{cm}} \ ,$$

$\vec{p}_{1\text{cm}}$ and $\vec{p}_{2\text{cm}}$ being the momenta of the two particles in the c.m.-frame.
This leads to

$$\vec{\beta} = \frac{\vec{p}_{1\text{lab}} + \vec{p}_{2\text{lab}}}{E_{1\text{lab}} + E_{2\text{lab}}} \ . \tag{B.5}$$

In tpeql the sum of the lab energies is stored, in tpbeta the vector $\vec{\beta}$, tpbq is set to β^2.

```
c
c Calculate beta-vector for transformation from lab-frame to cm.
      tpeql   = eq(tprn,tpp1)  +eq(tprn,tpp2)
      tpbeta(1)=(pq(tprn,tpp1,1)+pq(tprn,tpp2,1))/tpeql
      tpbeta(2)=(pq(tprn,tpp1,2)+pq(tprn,tpp2,2))/tpeql
      tpbeta(3)=(pq(tprn,tpp1,3)+pq(tprn,tpp2,3))/tpeql
c Calculate beta squared
      tpbq=tpbeta(1)**2+tpbeta(2)**2+tpbeta(3)**2
```

For the total energy squared in the c.m.-frame tpsq equation (B.3) is used. There the vector $\vec{p}_{\rm cm}$ is the total momentum of the two particles in the c.m.-frame and therefore zero.

Division by γ results in

$$S^2 := E_{\rm cm}^2 = \frac{E_{\rm lab}^2}{\gamma^2} = E_{\rm lab}^2(1-\beta^2)\ . \tag{B.6}$$

```
c Calculate total energy c.m.
      tpsq=tpeql*tpeql*(1.-tpbq)
```

A few statistics are set for the total energy squared:

```
c Statistics
c
      if (tpsq.lt.minsq) minsq=tpsq
      if (tpsq.gt.maxsq) maxsq=tpsq
```

Further tests in the lab frame

After the distance in the lab frame had been compared to the maximum assumed cross section now the comparision to the real cross section for the process is performed. The index tpcrindex is used for looking up the cross section from the cross section arrays, see subroutine initcross in B.3.7 on page 106.

sqstep is set in stdec, so is maxpoints as the maximum lookup-table index.

The rounding is done by truncation. This is important as otherwise it could happen that if the energy-limit below which the collision is kinematically forbitten is just between the two indizes in question, and rounding up leads to a forbidden collision.

```
c
c Calculate index for lookup table
c
      tpcrindex=int(sqrt(sqrt((tpsq-eps)/sqstep)))+1
      if (tpcrindex.gt.maxpoints) tpcrindex=maxpoints
c
```

The variables tin1 and tin2 are set to the particle types of the incoming particles:

```
      tin1=tq(tprn,tpp1)
      tin2=tq(tprn,tpp2)
```

The total cross section tpsigma is looked up from the cross section tables, also the incoming channel tpchin is set:

```
c Get total cross section
      call totalcross()
```

The statistics for minsigma and maxsigma are set:

```
c
c Statistics
c
      if (tpsigma.lt.minsigma) minsigma=tpsigma
      if (tpsigma.gt.maxsigma) maxsigma=tpsigma
```

Now the distance is compared to the real cross section:

```
c
c ==========================================================================
c 2nd check                                                              !!
c ---------                                                              !!
      if (tpdq.gt.tpsigma/pi) return                                    !!
c Increase statistics: Passed second test in this test run               !!
      nsta2(tprn)=nsta2(tprn)+1                                          !!
c ==========================================================================
c
```

The transformation of the momenta to the c.m.-system

tpgamma is set to $\gamma = \frac{1}{\sqrt{1-\beta^2}}$. For the transformation the values of $\frac{\gamma-1}{\beta^2}$ and $\vec{\beta}\vec{p}_{1\text{lab}}$ are calculated in tpgambet and scalbetax respectively.

In case the two paricles already are nearly in their c.m.-system ($|\beta| \ll 1$), the non-relativistic limit of the transformation is used:

```
c Now for the transformation to c.m. system!
c -----------------------------------------
c
c Calculate gamma
      tpgamma=1./sqrt(1.-tpbq)
c
      if (tpbq.gt.eps) then
         tpgambet=(tpgamma-1.)/tpbq
c Transform momentum 1
         scalbetax=tpbeta(1)*pq(tprn,tpp1,1)+
     &              tpbeta(2)*pq(tprn,tpp1,2)+
     &              tpbeta(3)*pq(tprn,tpp1,3)
```

The variable a is only an appreviation and can be used in equation (B.2) when $\vec{\beta}$ is written in front of the two last terms:

```
         a=tpgambet*scalbetax-tpgamma*eq(tprn,tpp1)
```

Now for equations (B.1)...

```
c
         tpeq1   =tpgamma*(eq(tprn,tpp1)-scalbetax)
```

...and (B.2):

```
         tpmo1(1)=pq(tprn,tpp1,1)+a*tpbeta(1)
         tpmo1(2)=pq(tprn,tpp1,2)+a*tpbeta(2)
         tpmo1(3)=pq(tprn,tpp1,3)+a*tpbeta(3)
c
```

Now the same transformation could be done for particle 2, but the result of the momentum transformation is clear: $-\vec{p}_{2\text{cm}}$.

```
c Transform momentum 2
         scalbetax=tpbeta(1)*pq(tprn,tpp2,1)+
     &             tpbeta(2)*pq(tprn,tpp2,2)+
     &             tpbeta(3)*pq(tprn,tpp2,3)
c
         tpeq2   =tpgamma*(eq(tprn,tpp2)-scalbetax)
```

The momentum of course is not transformed.

The else branch of the condition does a non-relativistic transformation:

```
      else
c
c Nonrelativistic transformation as limit for low beta
c
         scalbetax=0.
         tpeq1   =eq(tprn,tpp1)
         tpmo1(1)=(pq(tprn,tpp1,1)-pq(tprn,tpp2,1))/2.
         tpmo1(2)=(pq(tprn,tpp1,2)-pq(tprn,tpp2,2))/2.
         tpmo1(3)=(pq(tprn,tpp1,3)-pq(tprn,tpp2,3))/2.
         tpeq2   =eq(tprn,tpp2)
      endif
c
```

A small check is done to the transformation by examining if the sum of the two particle energies really equals the total energy previously calculated. If not, the program is aborted with a report written to the protocol file:

```
c ----------------------------------------------------------------------
c Small check                                                          !
      if (abs((tpeq1+tpeq2)**2-tpsq)/tpsq.gt.0.01) then                !
         write(1,*) 'ERROR in Lorentztransformation:'                  !
         write(1,*) (tpeq1+tpeq2)**2,tpsq                              !
         write(1,*) '1 : ',                                            !
     &    eq(tprn,tpp1),pq(tprn,tpp1,1),pq(tprn,tpp1,2),pq(tprn,tpp1,3) !
         write(1,*) 'M1: ',                                            !
     &    sqrt(eq(tprn,tpp1)**2-pq(tprn,tpp1,1)**2                     !
     &    -pq(tprn,tpp1,2)**2-pq(tprn,tpp1,3)**2)                      !
         write(1,*) '2: ',                                             !
     &    eq(tprn,tpp2),pq(tprn,tpp2,1),pq(tprn,tpp2,2),pq(tprn,tpp2,3) !
         write(1,*) 'M2: ',                                            !
     &    sqrt(eq(tprn,tpp2)**2-pq(tprn,tpp2,1)**2                     !
     &    -pq(tprn,tpp2,2)**2-pq(tprn,tpp2,3)**2)                      !
         write(1,*) 'tpeq1/2: ',tpeq1,tpeq2                            !
         write(1,*) 'tpmo1  :'                                         !
         write(1,*) tpmo1(1),tpmo1(2),tpmo1(3)                         !
         write(1,*) 'M1t:',                                            !
     &              sqrt(tpeq1**2-tpmo1(1)**2-tpmo1(2)**2-tpmo1(3)**2) !
         write(1,*) 'M2t:',                                            !
     &              sqrt(tpeq2**2-tpmo1(1)**2-tpmo1(2)**2-tpmo1(3)**2) !
         stop 'ERROR in Lorentztransformation'                         !
      endif                                                            !
c ----------------------------------------------------------------------
```

The transformation of the relative position to the c.m.-system

The distance vector of the two particles is also transformed with equations (B.1) and (B.2) used for the position rather than the momentum. The time difference between the two particles in the lab frame is zero. The time distance in the c.m.-frame is called `tptd`, the spatial distance `tprc`.

Again it has to be checked if a non-relativistic transformation should be used:

```
c
c Transform relative position
      if (tpbq.gt.eps) then
         scalbetax=tpbeta(1)*tprl(1)+
     &             tpbeta(2)*tprl(2)+
     &             tpbeta(3)*tprl(3)
         a=tpgambet*scalbetax
c
         tptd   =tpgamma*scalbetax
         tprc(1)=tprl(1)+a*tpbeta(1)
         tprc(2)=tprl(2)+a*tpbeta(2)
         tprc(3)=tprl(3)+a*tpbeta(3)
```

In case no transformation is necessary the values are set directly:

```
      else
         scalbetax=0.
         tptd=0.
         tprc(1)=tprl(1)
         tprc(2)=tprl(2)
         tprc(3)=tprl(3)
      endif
```

The transformation to the c.m.-frame is somewhat strange if afterwards the time distance between the two particles is bigger than the transformed timestep length, called `cmtimstp`. `cmtimstp` is calculated now by means of (B.3)...

```
c
c Transform timestep-length
      cmtimstp=timstp/tpgamma
```

...and compared to the time distance. If the time distance is bigger, the statistics counter `timefault` is incremented:

```
c Time difference in c.m. frame > timestep length?
      if (abs(tptd).gt.cmtimstp) timefault=timefault+1
```

The spatial distance squared is calculated and named `tpdqc`:

```
c Distance in c.m. frame
      tpdqc=tprc(1)**2+tprc(2)**2+tprc(3)**2
c
```

Checks in the c.m.-frame

The condition 'c.m.-distance greater than cross section radius' throws out more events than the condition 'impact parameter greater than cross section radius'! Therefore it is more effective and still correct because in the case when there really is scattering both conditions are equivalent – as already pointed out because of the second condition 'closest approach' the impact parameter equals the distance.

```
c ===========================================================================
c 3rd check                                                                !!
c ---------                                                                !!
c The condition 'distance greater than cross section' throws out           !!
c more events than the condition 'impact parameter greater than            !!
c cross section'! Therefore it's more effective.                           !!
c In the case when there is really scattering both conditions are          !!
c equivalent since 'impact parameter=distance' for closest approach.       !!
c                                                                          !!
      if (tpdqc.gt.tpsigma/pi) return                                      !!
c Increase statistics: Passed third test in this test run                  !!
      nsta3(tprn)=nsta3(tprn)+1                                            !!
c ===========================================================================
```

Now the 'closest approach' test is performed.

It is done by calculating the scalar product of the distance vector with the momentum vector both now and one timestep later. At the point of closest approach itself the scalar product is zero. However the changing of the sign of the scalar product from now to the next timestep is taken as an indication that the point of closest approach will be reached in this timestep.

The first scalar product can be calculated directly:

```
c
c Closest approach test
c ---------------------
c Calculate scalar product of c.m.-distance and c.m.-momentum
c (proportional to angle between them!)
      rpcm1=tprc(1)*tpmo1(1)+tprc(2)*tpmo1(2)+tprc(3)*tpmo1(3)
```

This is followed by the propagation of the partons in the lab-frame and subsequent transformation of the new positions to the c.m.-frame. The propagation vectors for the partons take the form

$$\left(\Delta t_{\text{step}}, \frac{\boldsymbol{p}_{\text{lab,i}}}{E_{\text{lab,i}}}\Delta t_{\text{step}}\right) .$$

With $\boldsymbol{\Lambda}_{a_0}(\boldsymbol{a})$ being the spatial Lorentz-transformation into the c.m.-frame for a vector $(a_0, \boldsymbol{a})$, one gets

$$\begin{aligned}\Delta \boldsymbol{r}'_{\text{cm}} &= \boldsymbol{\Lambda}_{0+\Delta t_{\text{step}}-0-\Delta t_{\text{step}}}\left(\boldsymbol{r}_{\text{lab},1} + \frac{\boldsymbol{p}_{\text{lab},1}}{E_{\text{lab},1}}\Delta t_{\text{step}} - \boldsymbol{r}_{\text{lab},2} - \frac{\boldsymbol{p}_{\text{lab},2}}{E_{\text{lab},2}}\Delta t_{\text{step}}\right) \qquad \text{(B.7)}\\ &= \Delta \boldsymbol{r}_{\text{cm}} + \boldsymbol{\Lambda}_0\left(\frac{\boldsymbol{p}_{\text{lab},1}}{E_{\text{lab},1}}\Delta t_{\text{step}} - \frac{\boldsymbol{p}_{\text{lab},2}}{E_{\text{lab},2}}\Delta t_{\text{step}}\right)\\ &= \Delta \boldsymbol{r}_{\text{cm}} + \boldsymbol{\Lambda}_0\left(\frac{\boldsymbol{p}_{\text{lab},1}}{E_{\text{lab},1}}\Delta t_{\text{step}}\right) - \boldsymbol{\Lambda}_0\left(\frac{\boldsymbol{p}_{\text{lab},2}}{E_{\text{lab},2}}\Delta t_{\text{step}}\right) .\end{aligned}$$

Because of

$$\Lambda_0(\boldsymbol{p}_{\rm lab,i}) = \Lambda_{E_{\rm lab,i}}(\boldsymbol{p}_{\rm lab,i}) + \gamma\beta E_{\rm lab,i} = \boldsymbol{p}_{\rm cm,i} + \gamma\beta E_{\rm lab,i} \tag{B.8}$$

the result is

$$\Delta \boldsymbol{r}'_{\rm cm} \;=\; \Delta \boldsymbol{r}_{\rm cm} + \left(\frac{\boldsymbol{p}_{\rm cm}}{\gamma(E_{\rm cm,1} + \beta \boldsymbol{p}_{\rm cm})} + \frac{\boldsymbol{p}_{\rm cm}}{\gamma(E_{\rm cm,2} - \beta \boldsymbol{p}_{\rm cm})}\right) \Delta t_{\rm step} \;. \tag{B.9}$$

```
c Calculate distance vector one timestep later
c
      dvcm(1)=tprc(1)+
     &        (tpmol(1)/eq(tprn,tpp1)+tpmol(1)/eq(tprn,tpp2))*timstp
      dvcm(2)=tprc(2)+
     &        (tpmol(2)/eq(tprn,tpp1)+tpmol(2)/eq(tprn,tpp2))*timstp
      dvcm(3)=tprc(3)+
     &        (tpmol(3)/eq(tprn,tpp1)+tpmol(3)/eq(tprn,tpp2))*timstp
c Calculate same scalarproduct as before one timestep later
      rpcm2=dvcm(1)*tpmol(1)+dvcm(2)*tpmol(2)+dvcm(3)*tpmol(3)
```

The closest approach test is now simply done by checking the sign of the product of the two scalar products:

```
c
c ==========================================================================
c 4th check                                                               !!
c ---------                                                               !!
      if (rpcm1*rpcm2.gt.0.) return                                       !!
c Increase statistics: Passed forth test in this run                      !!
      nsta4(tprn)=nsta4(tprn)+1                                           !!
c ==========================================================================
```

If the particles have passed all this test, they will scatter:

Scattering the particles

```
c
c ===========================================================================
c Okay, scatter
c ===========================================================================
```

But first it is made sure that the two particles really exist:

```
c
c -------------------------------------------------------------------------
c Last check: 'last' check                                                 !
      if ((last(tprn,tpp1).eq.-1).or.(last(tprn,tpp2).eq.-1)) then         !
         write(1,*) 'Trying to collide non-existing particles ...'         !
         write(1,*) 'tprn:',tprn                                           !
         write(1,*) 'tpp1:',tpp1,' last:',last(tprn,tpp1),' next:',        !
     &              nextgrid(tprn,tpp1)                                    !
```

```
          write(1,*) 'tpp2:',tpp2,' last:',last(tprn,tpp2),' next:',       !
     &              nextgrid(tprn,tpp2)                                    !
          write(1,*) 'nq(tprn):',nq(tprn)                                  !
          call checknumbers(tprn,tpp1,tpp2)                                !
          stop 'Trying to collide non-existing particles ...'              !
        endif                                                              !
c ------------------------------------------------------------------------
```

Now the output channel is chosen by subroutine outchannel, see B.8.2 on page 132. The channel number is stored in tpchout afterwards:

```
c
c Determine output channel tpcrout
        call outchannel()
c
```

The subroutine xsecentry (see B.8.4, page 136) is called to check the cross section entry in the arrays xsec...:

```
c ------------------------------------------------------------------------
c Check cross section table                                                !
        call xsecentry()                                                   !
c ------------------------------------------------------------------------
```

Since at the moment only $2 \to 2$ processes are implemented, the subroutine twototwo is directly called to complete this process. However, once $2 \to 1$ processes are implemented, depending on tpchout another routine could be called, too.

```
c
        call twototwo()
c
        return
        end
```

B.8.2 Subroutine outchannel

The subroutine outchannel is responsible for choosing the output channel based on the relative sizes of the partial cross sections:

```
c
c ---------------------------------------------------------------------
c Determines outgoing particles
c
        subroutine outchannel()
c
        include 'stdec.for'
c Random number
        double precision rd
c Do variable
        integer i
```

The following reminds the channel numbers as descripted in B.1.8 on page 91:

```
c
c Channels:
c              1 - gg
c              2 - qg
c              3 - qq
c              4 - qqb
c              5 - qqp
c              6 - qpqbp
```

The following gives an overview of the cross section arrays:

```
c
c>>      common /cross/ xsecgg22(maxpoints),
c>>     &               xsecqq22(maxtype,maxpoints),
c>>     &               xsecqg22(maxtype,maxpoints),
c>>     &               xsecqqb22(maxtype,maxpoints),
c>>     &               xsecqqp22(maxtype,maxtype,maxpoints),
c>>     &               xsecqqbgg(maxtype,maxpoints),
c>>     &               xsecggqqb(maxtype,maxpoints),
c>>     &               xsecqqbqpqbp(maxtype,maxtype,maxpoints)
```

For a description of this cross section arrays see B.3.7 on page 106.

A random number between 0 and the total cross section `tpsigma` is determined:

```
c
c Random number between 0 and total cross section
c
      rd=ran1(iseed)*tpsigma
c
```

The variable `tpchout` will be set by this routine according to the output channel number determined. It is preset to -1 to check later if a channel had been found:

```
      tpchout=-1
```

The open output channels of course depend on the input channel, stored in `tpchin` and determined by the subroutine `totalcross`, see B.8.3 on page 135:

```
c
      if (tpchin.eq.1) then
```

Incoming channel 1 is gg, so the open output channels are gg or $q\bar{q}$. The partial cross sections for two incoming gluons are added up in the order $gg \to gg$, $gg \to u\bar{u}$, $gg \to d\bar{d}$, ..., $gg \to t\bar{t}$:

```
         if (rd.le.xsecgg22(tpcrindex)) then
            tpchout=1
            tout1=tin1
            tout2=tin2
         else
            do 10 i=1,maxtype
               if (rd.le.xsecggqqb(i,tpcrindex)) then
                  tpchout=4
                  tout1=i
                  tout2=-i
                  goto 30
               endif
10          continue
         endif
      endif
```

Label 30 is just before final checks and exiting the routine. The variables tin1 and tin2 are set to the particle types of the incoming partons, tout1 and tout2 are to be set by this routine to the parton types of the outgoing partons.

Incoming channel 2 is qg, only the elastic channel is open:

```
c
      if (tpchin.eq.2) then
         tpchout=2
         tout1=tin1
         tout2=tin2
      endif
```

Incoming channel 3 is qq, again only the elastic channel is open:

```
c
      if (tpchin.eq.3) then
         tpchout=3
         tout1=tin1
         tout2=tin2
      endif
```

Incoming channel 4 is $q\bar{q}$, the partial cross sections are added up in the following order: $q\bar{q} \to q\bar{q}$, $q\bar{q} \to gg$, $q\bar{q} \to q'\bar{q}'$. The cross section array entries for $q\bar{q} \to q'\bar{q}'$ are added up in the order $q\bar{q} \to u\bar{u}$, $q\bar{q} \to d\bar{d}$ and set to zero when $q = q'$.

```
c
      if (tpchin.eq.4) then
         if (rd.le.xsecqqb22(abs(tin1),tpcrindex)) then
            tpchout=4
            tout1=tin1
            tout2=tin2
         else
            if (rd.le.xsecqqbgg(abs(tin1),tpcrindex)) then
               tpchout=1
               tout1=0
               tout2=0
            else
               do 20 i=1,maxtype
                  if (rd.le.xsecqqbqpqbp(abs(tin1),i,tpcrindex)) then
                     tpchout=6
                     tout1=i
                     tout2=-i
                     goto 30
                  endif
```

```
20              continue
            endif
         endif
      endif
```

For the incoming channel 5, qq', again only the elastic outchannel is open:

```
      if (tpchin.eq.5) then
         tpchout=5
         tout1=tin1
         tout2=tin2
      endif
```

Label 30 is the exit-label:

```
c
30    continue
c
```

As a small check it is made sure that actually one outchannel had been found:

```
      if (tpchout.lt.0.) then
         write(1,*) 'No output channel found'
         write(1,*) tin1,tin2,tpchin
         stop 'No output channel found'
      endif
c
      return
      end
```

B.8.3 Subroutine totalcross

This subroutine determines the total cross section from the cross section arrays. It is also responsible for determining the incoming channel `tpchin`. As the partial cross sections have been added up by the subroutine `initcross`, see B.3.7 on page 106, the total cross section is found in the entry for the last partial cross section in this sequence:

```
c ---------------------------------------------------------------------
c Determines total cross section
c
      subroutine totalcross()
c
c Channels:
c              1 - gg
c              2 - qg
c              3 - qq
c              4 - qqb
c              5 - qqp
c              6 - qpqbp
c
c
      include 'stdec.for'
c
```

The total cross section will be written to tpsigma, it is set to -1 first for check purposes:

```
      tpsigma=-1.
```

The variables tin1 and tin2 are set to the types of the incoming partons.

```
c
      if (tin1.eq.tin2) then
         if (tin1.eq.0) then
            tpsigma=xsecggqqb(maxtype,tpcrindex)
            tpchin=1
         else
            tpsigma=xsecqq22(abs(tin1),tpcrindex)
            tpchin=3
         endif
      else
         if (tin1.eq.-tin2) then
            tpsigma=xsecqqbqpqbp(abs(tin1),maxtype,tpcrindex)
            tpchin=4
         else
            if ((tin1.eq.0).or.(tin2.eq.0)) then
               tpsigma=xsecqg22(abs(tin1+tin2),tpcrindex)
               tpchin=2
            else
               tpsigma=xsecqqp22(abs(tin1),abs(tin2),tpcrindex)
               tpchin=5
            endif
         endif
      endif
```

As small checks the total cross section is compared to the maximum cross section maxsig being determined by the subroutine initcross, also, it is checked if the subroutine actually could determine the cross section:

```
c
      if (tpsigma.gt.maxsig) then
         write(1,*) 'maxsig not maximal'
         write(1,*) tin1,tin2,tpcrindex
         stop 'maxsig not maximal'
      endif
c
      if (tpsigma.lt.0.) then
         write(1,*) 'tpsigma not calculated'
         write(1,*) tin1,tin2
         stop 'tpsigma not calculated'
      endif
c
      return
      end
```

B.8.4 Subroutine xsecentry

The subroutine xsecentry checks the validity of the cross section tables by calculating the partial cross section for the given input and output channels and comparing them to the table entries. This subroutine again reflects the structure of the cross section arrays, in order to get a partial cross section for a process one has to subtract the previous entry in the cross section sequence from the given one.

For a description of this cross section arrays see B.3.7 on page 106.

```
c
c ---------------------------------------------------------------------
c Calculate cross section for given channels and compare to table
c
      subroutine xsecentry()
c
      include 'stdec.for'
c
c Channels:
c                 1 - gg
c                 2 - qg
c                 3 - qq
c                 4 - qqb
c                 5 - qqp
c                 6 - qpqbp
c
c TSIGGG22.FOR;1      TSIGGGQQB.FOR;1     TSIGQG22.FOR;1      TSIGQQ22.FOR;1
c TSIGQQB22.FOR;1     TSIGQQBGG.FOR;1     TSIGQQBQPQBP.FOR;1  TSIGQQP22.FOR;1
c
c>>         common /cross/ xsecgg22(maxpoints),
c>>       &                xsecqq22(maxtype,maxpoints),
c>>       &                xsecqg22(maxtype,maxpoints),
c>>       &                xsecqqb22(maxtype,maxpoints),
c>>       &                xsecqqp22(maxtype,maxtype,maxpoints),
c>>       &                xsecqqbgg(maxtype,maxpoints),
c>>       &                xsecggqqb(maxtype,maxpoints),
c>>       &                xsecqqbqpqbp(maxtype,maxtype,maxpoints)
c
      double precision tabsig
c
      if (tpchin.eq.tpchout) then
         if (tpchin.eq.1) then
            call tsiggg22()
            tabsig=xsecgg22(tpcrindex)
         endif
         if (tpchin.eq.2) then
            call tsigqg22()
            tabsig=xsecqg22(abs(tin1+tin2),tpcrindex)
         endif
         if (tpchin.eq.3) then
            call tsigqq22()
            tabsig=xsecqq22(abs(tin1),tpcrindex)
         endif
         if (tpchin.eq.4) then
            call tsigqqb22()
            tabsig=xsecqqb22(abs(tin1),tpcrindex)
         endif
         if (tpchin.eq.5) then
            call tsigqqp22()
            tabsig=xsecqqp22(abs(tin1),abs(tin2),tpcrindex)
         endif
      else
         if (tpchin.eq.1) then
            call tsigggqqb()
            if (abs(tout1).gt.1) then
               tabsig=xsecggqqb(abs(tout1)  ,tpcrindex)
     &               -xsecggqqb(abs(tout1)-1,tpcrindex)
            else
               tabsig=xsecggqqb(1,tpcrindex)-xsecgg22(tpcrindex)
            endif
         endif
         if (tpchin.eq.4) then
            if (tpchout.eq.1) then
               call tsigqqbgg()
               tabsig=xsecqqbgg(abs(tin1),tpcrindex)
     &               -xsecqqb22(abs(tin1),tpcrindex)
            endif
            if (tpchout.eq.6) then
               call tsigqqbqpqbp()
               if (abs(tout1).gt.1) then
                  tabsig=xsecqqbqpqbp(abs(tin1),abs(tout1)  ,tpcrindex)
     &                  -xsecqqbqpqbp(abs(tin1),abs(tout1)-1,tpcrindex)
               else
                  tabsig=xsecqqbqpqbp(abs(tin1),1,tpcrindex)
     &                  -xsecqqbgg(abs(tin1),tpcrindex)
               endif
            endif
         endif
      endif
c
      if (tpsigma.ne.0.) then
         if (abs(tpsigma-tabsig)/tpsigma.gt..1) then
            write(1,*)
     &  'Table entry for cross section off by more than 10 percent'
            write(1,*) 'tpsigma:',tpsigma,' tabsig:',tabsig
            write(1,*) 'tpsq:',tpsq,' tpcrindex ...:',
     &                         eps+(tpcrindex-1.)**4*sqstep
            write(1,*) 'tpchin/tpchout:',tpchin,tpchout
            write(1,*) 'tin1,2',tin1,tin2,'  tout1,2',tout1,tout2
            stop
     &  'Table entry for cross section off by more than 10 percent'
         endif
      else
         if (tabsig.gt.1.e-4) then
            write(1,*) 'Table entry for zero cross section .gt. 1.e-4'
            write(1,*) 'tpsigma:',tpsigma,' tabsig:',tabsig
            write(1,*) 'tpsq:',tpsq,' tpcrindex ...:',
     &                         eps+(tpcrindex-1.)**4*sqstep
            write(1,*) 'tpchin/tpchout:',tpchin,tpchout
            write(1,*) 'tin1,2',tin1,tin2,'  tout1,2',tout1,tout2
            stop 'Table entry for zero cross section .gt. 1.e-4'
         endif
      endif
```

```
c
      return
      end
```

B.9 TWOTOTWO

Gerd Kortemeyer

Last revision of file: 09/02/94

B.9.1 Subroutine twototwo

This subroutine completes the scattering of two particles into an output channel with also two particles. It is called by `collide`.

```
c
c Causal index, angle index, DO variable
      integer ic,ia,i
c Factors in transformation formula
      double precision scalbetax,a
c Middle point
      double precision midp(3)
c Velocities, radius
      double precision velo1,velo2,radius
c Old momenta and energies, new momentum squared, scalar product
      double precision ms1,ms2,me1,me2,m1(3),m2(3),nms1,nms2,scp
c Angle
      double precision labang
```

Now the cosine of the scattering angle θ `tpcos` is calculated by the appropriate routines. Setting `tpcos` to '1' is only done for safety reasons if the monte carlos are commented out for test purposes.

```
c Get tpcos=cos(theta) for scattering
c ---------------------------------
c Default: No scattering
      tpcos=1.
c
c Do the Monte Carlo
c
      call montecarlo()
c
```

A small check if the cosine is really between -1 and 1 is performed:

```
c ----------------------------------------------------------------------------
c Check cosine                                                                !
      if ((tpcos.ge.1).or.(tpcos.le.-1)) then                                 !
         write(1,*) 'Error in Monte Carlo, rien ne va plus.'                  !
         write(1,*) 'tpcos:',tpcos                                            !
         write(1,*) 'tpchin,tpchout:',tpchin,tpchout                          !
         stop 'Error in Monte Carlo, rien ne va plus.'                        !
      endif                                                                  !
c ----------------------------------------------------------------------------
```

The angular distribution statistics are set:

```
c
      ia=int(tpcos*3.)+4
      if (tpcos.lt.0) ia=ia-1
      anglestat(tpchout,ia)=anglestat(tpchout,ia)+1
```

Now the parton types can be set according to `tout1,2`:

```
c
c Set types for output
c --------------------
      tq(tprn,tpp1)=tout1
      tq(tprn,tpp2)=tout2
c
```

The scattering itself is then done by subroutine `newmomenta`:

```
c Scatter according to tpcos
c --------------------------
      call newmomenta()
```

The subroutine `newmomenta` places the new momentum for the first particle into the vector `tpsc1`
Some statistics for the scattering angle are set:

```
c
c Statistics
c
      if (tpcos.lt.mincos) mincos=tpcos
      if (tpcos.gt.maxcos) maxcos=tpcos
```

The backtransformations have to be done simply with the vector $-\vec{\beta}$ instead of $\vec{\beta}$. However, first a few values are stored for statistics and checks:

```
c
c Backtransformation
c ------------------
c
c Remember values for angle statistics and checks
c
      me1=eq(tprn,tpp1)
      me2=eq(tprn,tpp2)
      ms1=0.
      ms2=0.
      do 100 i=1,3
         m1(i)=pq(tprn,tpp1,i)
         ms1=ms1+m1(i)*m1(i)
         m2(i)=pq(tprn,tpp2,i)
         ms2=ms2+m2(i)*m2(i)
100   continue
c
      scalbetax=-tpbeta(1)*tpscl(1)
     &          -tpbeta(2)*tpscl(2)
     &          -tpbeta(3)*tpscl(3)
c Momentum 1
      a=-tpgambet*scalbetax+tpgamma*tpsceq1
c
      eq(tprn,tpp1)  =tpgamma*(tpsceq1-scalbetax)
      pq(tprn,tpp1,1)=tpscl(1)+a*tpbeta(1)
      pq(tprn,tpp1,2)=tpscl(2)+a*tpbeta(2)
      pq(tprn,tpp1,3)=tpscl(3)+a*tpbeta(3)
c Momentum 2
      a=tpgambet*scalbetax+tpgamma*tpsceq2
c
      eq(tprn,tpp2)  =tpgamma*(tpsceq2+scalbetax)
      pq(tprn,tpp2,1)=-tpscl(1)+a*tpbeta(1)
      pq(tprn,tpp2,2)=-tpscl(2)+a*tpbeta(2)
      pq(tprn,tpp2,3)=-tpscl(3)+a*tpbeta(3)
```

The new lab momentum squared is calculated for various purposes:

```
c Calculate new momenta squared
      nms1=0.
      nms2=0.
      do 300 i=1,3
         nms1=nms1+pq(tprn,tpp1,i)*pq(tprn,tpp1,i)
         nms2=nms2+pq(tprn,tpp2,i)*pq(tprn,tpp2,i)
300   continue
c ----------------------------------------------------------------------------
```

The transformation is checked now by again calculating the total energy:

```
c Check transformation                                                      !
      tpeql    = eq(tprn,tpp1)  +eq(tprn,tpp2)                              !
      tpbeta(1)=(pq(tprn,tpp1,1)+pq(tprn,tpp2,1))/tpeql                     !
      tpbeta(2)=(pq(tprn,tpp1,2)+pq(tprn,tpp2,2))/tpeql                     !
      tpbeta(3)=(pq(tprn,tpp1,3)+pq(tprn,tpp2,3))/tpeql                     !
      tpbq=tpbeta(1)**2+tpbeta(2)**2+tpbeta(3)**2                           !
c                                                                           !
      if (abs(tpsq-tpeql*tpeql*(1.-tpbq))/tpsq.gt.0.001) then               !
         write(1,*) 'ERROR in Backtransformation or Scattering:'            !
         write(1,*) tpsq,tpeql*tpeql*(1.-tpbq)                              !
         write(1,*) eq(tprn,tpp1),pq(tprn,tpp1,3)                           !
         write(1,*) eq(tprn,tpp2),pq(tprn,tpp2,3)                           !
         stop 'ERROR in Backtransformation or Scattering'                   !
      endif                                                                 !
c                                                                           !
```

Also, it is checked if the parton had gotten to large an 'imaginary mass' during the transformation. 'Imaginary masses' occur due to numeric precision problems if the parton had a very little mass before the transformation which during the transformation calculations turned into a very little imaginary mass:

```
      if (eq(tprn,tpp1)*eq(tprn,tpp1)-nms1.lt.-.05) then               !
         write(1,*) 'Particle has non-negligible imaginary mass.'      !
         write(1,*) 'Old/new energy:',me1,eq(tprn,tpp1)                !
         write(1,*) 'Old/new momentum:'                                !
         write(1,*) m1(1),m1(2),m1(3)                                  !
         write(1,*) pq(tprn,tpp1,1),pq(tprn,tpp1,2),pq(tprn,tpp1,3)    !
         write(1,*) 'Old real mass :',sqrt(me1*me1-ms1)                !
         write(1,*) 'Imaginary mass:',                                 !
     &           sqrt(-eq(tprn,tpp1)*eq(tprn,tpp1)+nms1)               !
         write(1,*) 'tpsq :',tpsq                                      !
         write(1,*) 'tpcos:',tpcos                                     !
         stop 'Particle has non-negligible imaginary mass.'            !
      endif                                                            !
c                                                                      !
      if (eq(tprn,tpp2)*eq(tprn,tpp2)-nms2.lt.-.05) then               !
         write(1,*) 'Particle has non-negligible imaginary mass.'      !
         write(1,*) 'Old/new energy:',me2,eq(tprn,tpp2)                !
         write(1,*) 'Old/new momentum:'                                !
         write(1,*) m2(1),m2(2),m2(3)                                  !
         write(1,*) pq(tprn,tpp2,1),pq(tprn,tpp2,2),pq(tprn,tpp2,3)    !
         write(1,*) 'Old real mass :',sqrt(me2*me2-ms2)                !
         write(1,*) 'Imaginary mass:',                                 !
     &           sqrt(-eq(tprn,tpp2)*eq(tprn,tpp2)+nms2)               !
         write(1,*) 'tpsq :',tpsq                                      !
         write(1,*) 'tpcos:',tpcos                                     !
         stop 'Particle has non-negligible imaginary mass.'            !
      endif                                                            !
c -----------------------------------------------------------------------
c
```

After having made sure that the correction is negligible the parton is artificially given an eps-mass:

```
c Artificially correct imaginary masses. Above check makes sure that the
c correction is not too big. Particle given eps-mass.
c
      if (eq(tprn,tpp1)*eq(tprn,tpp1).lt.nms1)
     &    eq(tprn,tpp1)=sqrt(nms1)+eps
      if (eq(tprn,tpp2)*eq(tprn,tpp2).lt.nms2)
     &    eq(tprn,tpp2)=sqrt(nms2)+eps
```

The lab angle statistics are set:

```
c
c Lab angle statistics
c
      scp=0.
      do 200 i=1,3
         scp=scp+pq(tprn,tpp1,i)*m1(i)
200   continue
      labang=scp/sqrt(nms1*ms1)
      ia=int(labang*3.)+4
      if (labang.lt.0) ia=ia-1
      labangle(ia)=labangle(ia)+1
```

```
c
      scp=0.
      do 210 i=1,3
         scp=scp+pq(tprn,tpp2,i)*m2(i)
210   continue
      labang=scp/sqrt(nms2*ms2)
      ia=int(labang*3.)+4
      if (labang.lt.0) ia=ia-1
      labangle(ia)=labangle(ia)+1
```

Now that the particles are successfully scattered the flags have to be set accordingly. First the total number of scattering events is increased by one, the channel statistics are updated:

```
c
c Particles successfully scattered, set flags
c ------------------------------------------
c Total number
      totscat=totscat+1
c Channel table
      chscat(tpchin,tpchout)=chscat(tpchin,tpchout)+1
```

The variables `fq` are set to 'not belonging to the nucleus of origin anymore', the flags `alr` are set to 'scattered' and in the particle data `last` is set to the respectively other scattering partner.

```
c Don't belong to certain nucleus anymore
      fq(tprn,tpp1)=0
      fq(tprn,tpp2)=0
c Particles scattered in this run
      alr(tprn,tpp1)=1
      alr(tprn,tpp2)=1
c Set last scattering partners
      last(tprn,tpp1)=tpp1
      last(tprn,tpp2)=tpp2
```

The middle point of the scattering is calculated in the lab frame:

```
c Calculate middle point of scattering
      midp(1)=(rq(tprn,tpp1,1)+rq(tprn,tpp2,1))/2.
      midp(2)=(rq(tprn,tpp1,2)+rq(tprn,tpp2,2))/2.
      midp(3)=(rq(tprn,tpp1,3)+rq(tprn,tpp2,3))/2.
```

Then information about the distance of the scattering event from the beam axis and signal velocities is calculated. The signal velocity can only be calculated if there had been a previous scattering event:

```
c Causal information
      radius=sqrt(midp(1)**2+midp(2)**2)
c Velocity information
      if (nnq(tprn,tpp1).eq.0) then
         velo1=sqrt( (midp(1)-lastpoint(tprn,tpp1,1))**2
     &              +(midp(2)-lastpoint(tprn,tpp1,2))**2
     &              +(midp(3)-lastpoint(tprn,tpp1,3))**2 ) /
     &             ( (ntime  - lasttime(tprn,tpp1))*timstp )
```

```
         else
            velo1=-1
         endif
c
         if (nnq(tprn,tpp2).eq.0) then
            velo2=sqrt( (midp(1)-lastpoint(tprn,tpp2,1))**2
     &                 +(midp(2)-lastpoint(tprn,tpp2,2))**2
     &                 +(midp(3)-lastpoint(tprn,tpp2,3))**2 ) /
     &               ( (ntime  - lasttime(tprn,tpp2))*timstp )
         else
            velo2=-1
         endif
```

After the first scattering event the partons are not considered part of a certain nucleon anymore. With **nnq** one can check that, '1' means 'belonging to a certain nucleon', '-1' means just created in a scattering process, '0' means 'already scattered itself'.

```
c Don't belong to certain nucleon anymore
         nnq(tprn,tpp1)=0
         nnq(tprn,tpp2)=0
```

Finally, the information about the last scattering point is stored to the particle data:

```
c Set lastpoint
         lasttime(tprn,tpp1)=ntime
         lasttime(tprn,tpp2)=ntime
c
         lastpoint(tprn,tpp1,1)=midp(1)
         lastpoint(tprn,tpp2,1)=midp(1)
         lastpoint(tprn,tpp1,2)=midp(2)
         lastpoint(tprn,tpp2,2)=midp(2)
         lastpoint(tprn,tpp1,3)=midp(3)
         lastpoint(tprn,tpp2,3)=midp(3)
```

The infimation about the scattering event is written to the scattering data file:

```
c
         if (scattering.ne.0) write(4,*)
     &       ntime*timstp,midp(3),radius,velo1,velo2,sqrt(tpsq)
c
         return
         end
c
```

B.9.2 Subroutine montecarlo

This procedure calls the monte carlo routine belonging to `tpchin` and `tpchout`:

```
c
c ----------------------------------------------------------------------
c Call right monte carlo
```

```
c
      subroutine montecarlo()
c
      include 'stdec.for'
c
c Channels:
c              1 - gg
c              2 - qg
c              3 - qq
c              4 - qqb
c              5 - qqp
c              6 - qpqbp
c
c TSIGGG22.FOR;1      TSIGGGQQB.FOR;1     TSIGQG22.FOR;1       TSIGQQ22.FOR;1
c TSIGQQB22.FOR;1     TSIGQQBGG.FOR;1     TSIGQQBQPQBP.FOR;1   TSIGQQP22.FOR;1
c MONTEGG.FOR;1       MONTEGGQQB.FOR;1    MONTEQG.FOR;1        MONTEQQ.FOR;14
c MONTEQQB.FOR;3      MONTEQQBGG.FOR;1    MONTEQQBP.FOR;3      MONTEQQP.FOR;11
c
      if (tpchin.eq.tpchout) then
         if (tpchin.eq.1) call montegg()
         if (tpchin.eq.2) call monteqg()
         if (tpchin.eq.3) call monteqq()
         if (tpchin.eq.4) call monteqqb()
         if (tpchin.eq.5) call monteqqp()
      else
         if (tpchin.eq.1) call monteggqqb()
         if (tpchin.eq.4) then
            if (tpchout.eq.1) call monteqqbgg()
            if (tpchout.eq.6) call monteqqbp()
         endif
      endif
c
      return
      end
```

B.10 NEWMOMENTA

Gerd Kortemeyer

Last revision of the file: 10/19/93

The subroutine newmomenta is responsible to rotate the momentum vectors in the c.m.-frame according to the given scattering angle $\cos\theta$ stored in tpcos.

B.10.1 Local variables

```
      double precision pxr(3),pxra,uperp(3)
      double precision pxpxr(3),pxpxra,vperp(3),eperp(3)
      double precision tpsin,tpmoa,scp,phi,siphi,cophi
```

B.10.2 Construction of a coordinate system

First the cross product of the distance vector and the momentum vector is calculated. This is turned into a unit vector perpendicular to the momentum vector:

```
c
c Calculate cross product of tprc and tpmol
c
      pxr(1)=tprc(2)*tpmol(3)-tprc(3)*tpmol(2)
      pxr(2)=tprc(3)*tpmol(1)-tprc(1)*tpmol(3)
      pxr(3)=tprc(1)*tpmol(2)-tprc(2)*tpmol(1)
c
c Calculate length of this vector ...
c
      pxra=sqrt(pxr(1)*pxr(1)+pxr(2)*pxr(2)+pxr(3)*pxr(3))
c
c ... and produce unit-vector
c
      uperp(1)=pxr(1)/pxra
      uperp(2)=pxr(2)/pxra
      uperp(3)=pxr(3)/pxra
```

Now another unitvector is calculated which is perpendicular to both the first unit vector and the momentum vector. It is in the plane of the momentum vector and the distance vector, and points into the direction of the distance vector.

```
c
c Calculate cross product of tpmol and pxr.
c This vector is in the (tprc,tpmol) plane and perpendicular to tpmol,
c pointing away from tprc
c
      pxpxr(1)=tpmol(2)*pxr(3)-tpmol(3)*pxr(2)
      pxpxr(2)=tpmol(3)*pxr(1)-tpmol(1)*pxr(3)
      pxpxr(3)=tpmol(1)*pxr(2)-tpmol(2)*pxr(1)
c
c Calculate length of this vector ...
c
      pxpxra=sqrt(pxpxr(1)*pxpxr(1)+pxpxr(2)*pxpxr(2)+pxpxr(3)*pxpxr(3))
c
c ... and produce unit-vector
c
      vperp(1)=pxpxr(1)/pxpxra
      vperp(2)=pxpxr(2)/pxpxra
      vperp(3)=pxpxr(3)/pxpxra
```

Now with uperp and vperp a coordinate system for the rotation is provided.

B.10.3 Getting the angle φ

The angle φ is chosen randomly between 0 and 2π. `siphi` and `cophi` are used as appreviations for $\sin\varphi$ and $\cos\varphi$ respectively:

```
c
c Get phi randomly between 0 and 2pi
c
      phi=2.*pi*ran1(iseed)
c
      siphi=sin(phi)
      cophi=cos(phi)
```

Now with uperp and vperp as coordinate system a vector eperp which is perpendicular to tpmo1, but inclosing the angle φ with vperp, is constructed:

```
c
c Produce unit-vector with angle phi
c
      eperp(1)=siphi*uperp(1)+cophi*vperp(1)
      eperp(2)=siphi*uperp(2)+cophi*vperp(2)
      eperp(3)=siphi*uperp(3)+cophi*vperp(3)
```

Calculate the sine of the scattering angle θ:

```
c
c Calculate sin(theta)
c
      tpsin=sqrt(1-tpcos*tpcos)
```

Calculate the length of the momentum vector:

```
c
c Calculate length of momentum vector
c
      tpmoa=sqrt(tpmo1(1)*tpmo1(1)+tpmo1(2)*tpmo1(2)+tpmo1(3)*tpmo1(3))
```

Now everything is prepared for the rotation of the momentum vector. The rotated vector is called tpsc1:

```
c
c Rotate momentum theta degrees in eperp-direction
c
      tpsc1(1)=tpmo1(1)*tpcos+tpmoa*eperp(1)*tpsin
      tpsc1(2)=tpmo1(2)*tpcos+tpmoa*eperp(2)*tpsin
      tpsc1(3)=tpmo1(3)*tpcos+tpmoa*eperp(3)*tpsin
```

Finally it is checked if the angle between tpmo1 and tpsc1 is really θ degrees by calculating the scalar product of them:

```
c
c Check rotation
c
      scp=tpsc1(1)*tpmo1(1)+tpsc1(2)*tpmo1(2)+tpsc1(3)*tpmo1(3)
      if (abs(scp/(tpmoa*tpmoa)-tpcos).gt.0.01) then
         write(1,*) 'ERROR in rotation of momenta'
         write(1,*) 'MO: ',tpmo1(1),tpmo1(2),tpmo1(3)
         write(1,*) 'SC: ',tpsc1(1),tpsc1(2),tpsc1(3)
         write(1,*) scp/(tpmoa*tpmoa),tpcos
         stop
      endif
```

For this elastic scattering the energies didn't change, so the energies of the scattered particles `tpeqsc1` and `tpeqsc2` are set to the old values:

```
c
c Energies didn't change
c
      tpsceq1=tpeq1
      tpsceq2=tpeq2
c
      return
      end
```

B.11 DECAY

Gerd Kortemeyer

Last revision of the file: 01/31/94

This subroutine simulates the decay of particles. Besides `move` and `scatter` it is one of the main routines of the program. See A.6 on page 83 for a description.

```
      subroutine decay()
```

B.11.1 Local variables

```
c DO variables
      integer test,i
c Difference from on-shell mass times timestep length
      double precision ddx
c Function for on-shell masses
      double precision mshell
```

B.11.2 Decaying of the particles

The subroutine checks all testruns and particles:

```
c All testruns, all particles
      do 10 test=1,ntr
         do 10 i=1,nq(test)
```

Particles in question are – of course – existing, they have not collided in the `scatter` step and don't belong to a certain nucleon:

```
c Not deleted, not already collided and not hadronized
            if ((last(test,i).ne.-1).and.(alr(test,i).ne.1).and.
     &          (nnq(test,i).le.0)                              ) then
```

The variable ddx represents the value of Δmt_{step}:

```
c Calculate difference from on-shell mass times timestep length
               ddx=abs(sqrt( eq(test,i)  *eq(test,i)
     &                      -pq(test,i,1)*pq(test,i,1)
     &                      -pq(test,i,2)*pq(test,i,2)
     &                      -pq(test,i,3)*pq(test,i,3) )
     &                  - mshell(tq(test,i))                ) * timstp
```

The function mshell(tq(...)) provides the on-shell masses of the different parton types.

Now a random number is calculated and compared to the probability for decay, see A.6 on page 83. If it is smaller, the subroutine delpar (see B.12.1 on page 149) is called to delete the particle:

```
c Delete particle?
               if (ran1(iseed).lt.(ddx/(1.+ddx))) call delpar(test,i)
```

Now two other partons should be created, however, the theory for this is not yet worked out.

```
            endif
10    continue
c
      return
      end
```

B.12 CREDEL

Gerd Kortemeyer

Last revision of the file: 02/25/94

The subroutines in this file are responsible for creating and deleting particles. Also, a routine to check the validity of the particles data structure is included.

B.12.1 delpar

This subroutine is to be used when a particle should be deleted.

```
      subroutine delpar(test,n)
```

The two parameters test and n, both integer, specify the testrun and the particle number respectively.

The local variables are

```
c DO variables
      integer i,j,k
c Highest existing particle index
      integer highexpar
c Found particle in nextgrid chain?
      integer found
```

First it is checked if the particle to be deleted exists. This is done by means of the `last`-flag, see B.1.3 on page 87.

```
      if (last(test,n).eq.-1) then
         write(1,*) 'Trying to delete non-existing particle.'
         write(1,*) 'test:',test,' n:',n
         call checknumbers(test,n,0)
         stop 'Trying to delete non-existing particle.'
      endif
```

The procedure **checknumbers** is described in B.12.3 on page 154.

In the next step the particle is deleted:

```
c
      last(test,n)=-1
c
```

Instead of just marking the particle 'removed' and then during `scatter` always to check if the chosen particles exist it is faster to completely remove the particle from the **nextgrid** pointer structure. This is simply done by having the previous particle in the chain point to the subsequent one. The number of times the particle was found in the chain is counted in `found`. The result can only be '1' if it was inside of the chain, or '0' if it was a `gridroot`.

In addition to that in this loop the highest occupied array index is determined:

```
c Skip particle by removing it from the nextgrid chain.
c Find highest array index.
c
      found=0
      highexpar=-1
c
      do 10 i=1,nq(test)
         if (last(test,i).ne.-1) then
            if (nextgrid(test,i).eq.n) then
               nextgrid(test,i)=nextgrid(test,n)
               found=found+1
            endif
            highexpar=i
         endif
10    continue
```

Now if the loop was executed without finding the particle it must have been a gridroot. If so, the `gridroot` is set to the subsequent particle:

```
c If the loop was executed without finding the particle ...
c
      if (found.eq.0) then
c
c ... maybe it was a gridroot?
c
         do 30 i=-mgridxy,mgridxy
            do 30 j=-mgridxy,mgridxy
               do 30 k=-mgridz,mgridz
```

```
                  if (gridroot(test,i,j,k).eq.n) then
                     gridroot(test,i,j,k)=nextgrid(test,n)
                     found=1
                     goto 20
                  endif
30          continue
```

If the particle wasn't a gridroot either there is a serious problem. One might think that it still could be possible that the respective array position was deleted and than on the same position another particle could have been created in the same timestep – this particle would not be member of any particle chain, see B.12.2 on page 152. But a freshly created particle can not be deleted in the same timestep because for this timestep it by definition has already interacted, so:

```
c Still searching? Well, there is a serious problem!
c
            write(1,*) 'Particle to be deleted not found in nextgrid chain'
            write(1,*) 'Testrun',test,', particle',n
            write(1,*) 'nq(test):',nq(test)
            stop 'Particle to be deleted not found in nextgrid chain'
```

The next `endif` belongs to the `found.eq.0` condition.

```
         endif
```

The next label is the one it is jumped to when the particle was found to be a `gridroot`. The program is also running over this label when the particle was found as a `nextgrid`:

```
20       continue
```

The next run-time check is if the particle was found more than one time in the chains:

```
         if (found.ne.1) then
            write(1,*) 'Corrupted nextgrid chain.'
            write(1,*) 'Particle',n,' test',test,' found',found,' times'
            stop 'Corrupted nextgrid chain.'
         endif
```

If no existing particle was found below `nq` there might be a problem with it:

```
         if (highexpar.lt.0) then
            write(1,*) 'nq(test) found corrupted while deleting particle'
            write(1,*) 'Testrun',test,', nq(test)',nq(test)
            stop 'nq(test) found corrupted while deleting particle'
         endif
```

As everything seems to be going pretty well, the highest particle index for this testrun is updated according to `highexpar`:

```
         nq(test)=highexpar
```

Finally, the number of deleted particles is updated:

```
c Statistics
c
      delnumber(test)=delnumber(test)+1
```

Notice that the nextgrid pointer of the deleted particle is not changed! This is important because otherwise things would get screwed up in subroutine scatter. The nextgrid of the deleted particle still points to a valid member of the chain. The only thing that happened to the deleted particle: It is not pointed to anymore.

```
      return
      end
```

B.12.2 crepar

This subroutine creates a particle with the information from the common newpart.

```
      subroutine crepar()
```

Only one local variable is defined:

```
c DO variable
      integer i
```

The procedure first searches for a free space in the particle arrays:

```
c Find empty entry
c
      do 10 i=1,mquark
         if (last(newtest,i).eq.-1) goto 20
10    continue
```

When the loop was fully executed, no free space was found:

```
c ... Array is full
c
      write(1,*) 'Could not create particle, stack capacity exceeded.'
      write(1,*) 'Testrun',newtest,', nq(newtest)',nq(newtest)
      stop 'Could not create particle, stack capacity exceeded.'
```

The label 20 is jumped to as soon as a free space is found:

```
20    continue
```

The free space could be a hole in the positions 1 to nq(test) or the first entry above nq(test). In the latter case nq has to be updated. However, the new value can at most differ by one from the old value:

```
      if (i.gt.nq(newtest)) then
         if (i-nq(newtest).ne.1) then
            write(1,*)
     &       'nq(newtest) found corrupted while creating particle'
            write(1,*) 'Testrun',newtest,', nq(newtest)',nq(newtest)
            stop 'nq(newtest) found corrupted while creating particle'
         endif
         nq(newtest)=i
      endif
```

The variable crenumber is updated:

```
      crenumber(newtest)=crenumber(newtest)+1
```

The position now is marked 'used' by setting the last scattering partner to 'itself' instead of '-1':

```
      last(newtest,i)=i
```

Now the remaining data from the common block newpart is written to the free position. The nnq entry is set to '-1' meaning 'not belonging to a certain nucleon'.

```
c Position
      rq(newtest,i,1)=newrq(1)
      rq(newtest,i,2)=newrq(2)
      rq(newtest,i,3)=newrq(3)
c Momentum
      pq(newtest,i,1)=newpq(1)
      pq(newtest,i,2)=newpq(2)
      pq(newtest,i,3)=newpq(3)
c Energy
      eq(newtest,i)=neweq
c Type
      tq(newtest,i)=newtq
c Nucleon number
      nnq(newtest,i)=-1
c Nucleus number for 'already scattered'
      fq(newtest,i)=0
```

The updating of nextgrid will be done in the next move step.

Notice that the nextgrid pointer of the created particle is not changed! This is important because it could very well happen that the created particle is on an array-position that was occupied by a particle that has been deleted in the same collision. If the nextgrid pointer would have been modified things would get screwed up in subroutine scatter. The nextgrid of the created particle still points to a valid member of the deleted particles chain. But it is not pointed to. The new particle must not be scattered in *this* timestep anyway, so...

```
      return
      end
```

B.12.3 checknumbers

This subroutine checks the validity of the particle data structures. It can also write information about two particles chosen by the parameters to the protocol file – this option can be used for debug purposes and had been very useful in the past.

```
      subroutine checknumbers(test,n1,n2)
```

The parameters are the testrun number and the particle numbers of the two particles to be checked. If the parameters are set to '0' no particles are checked:

```
c Parameters
      integer test,n1,n2
c Do variables
      integer i,j,k,l,m
c Count variables
      integer pcount, tcount,typecount(-6:6)
```

The procedure itself is coded pretty straightforward:

```
      do 10 i=1,ntr
c
c Count particles in data arrays
c
         pcount=0
c
         do 20 j=1,mquark
            if (last(i,j).gt.0) pcount=pcount+1
20       continue
c
c Count particles in pointer chain
c
         tcount=0
c
         do 25 j=-6,6
            typecount(j)=0
25       continue
c
         do 30 j=-mgridxy,mgridxy
            do 30 k=-mgridxy,mgridxy
               do 30 l=-mgridz,mgridz
c
                 m=gridroot(i,j,k,l)
                 if (m.eq.-1) goto 40
c
                 if ((m.eq.n1).and.(i.eq.test)) then
                    write(1,*) '------------------------------'
                    write(1,*) 'Particle ',n1,' is gridroot!'
                 endif
                 if ((m.eq.n2).and.(i.eq.test)) then
                    write(1,*) '------------------------------'
                    write(1,*) 'Particle ',n2,' is gridroot!'
                 endif
```

```
c
50                   tcount=tcount+1
                     typecount(tq(i,m))=typecount(tq(i,m))+1
                     if ((m.eq.n1).and.(i.eq.test)) then
                        write(1,*) '------------------------------'
                        write(1,*) 'Particle:',n1
                        write(1,*) 'Gridbox :',j,k,l
                     endif
                     if ((m.eq.n2).and.(i.eq.test)) then
                        write(1,*) '------------------------------'
                        write(1,*) 'Particle:',n2
                        write(1,*) 'Gridbox :',j,k,l
                     endif
c
                     m=nextgrid(i,m)
                     if (m.ne.-1) goto 50
c
40                   continue
30         continue
c
c Print out information
c
           write(1,*)
     &      'Number of particles cre/del: ',crenumber(i),delnumber(i)
           write(1,*)
     &      'Total number of particles  : ',pcount
           write(1,*)
     &      'Grid box chains            : ',tcount
           write(1,*) 'Gluons:',typecount(0)
           write(1,*) 'u, ubar:',typecount(1),typecount(-1)
           write(1,*) 'd, dbar:',typecount(2),typecount(-2)
           write(1,*) 's, sbar:',typecount(3),typecount(-3)
           write(1,*) 'c, cbar:',typecount(4),typecount(-4)
           write(1,*) 'b, bbar:',typecount(5),typecount(-5)
           write(1,*) 't, tbar:',typecount(6),typecount(-6)
c
c Right number of particles?
c
           if (lastnumber(i)+crenumber(i)-delnumber(i).ne.pcount) then
              write(1,*) 'Total number of particles incorrect.'
              write(1,*) 'pcount: ',pcount,', lastnumber',lastnumber(i)
              stop 'Total number of particles incorrect'
           endif
c
           lastnumber(i)=pcount
c
10     continue
c
       return
       end
```

B.13 PERMUTE

Gerd Kortemeyer

Last revision of the file: 12/7/93

This subroutine permutes the permutation arrays.

B.13.1 Local variables

Besides the variables from `stdec.for` the following local variables are defined:

```
c Main loop
      integer k
c Entries to be exchanged
      integer i,j
c Buffer
      integer exch
```

B.13.2 Permutation

The permutation of the respective arrays is done simply by exchanging *size of array* times two entries of the arrays.

First the array `perm` is permuted:

```
      do 10 k=1,mquark
c Find entry numbers
         i=1+int(ran1(iseed)*(mquark-1))
         j=1+int(ran1(iseed)*(mquark-1))
         if ((i.gt.mquark).or.(i.lt.1).or.(j.gt.mquark).or.(j.lt.1))
     &   then
            write(1,*) 'Error in random numbers, i,j=',i,j
            write(1,*) 'mquark:',mquark
            stop
         endif
c Exchange them
         exch=perm(i)
         perm(i)=perm(j)
         perm(j)=exch
c
10    continue
```

Now `permx` and `permy` are permuted. One has to be careful not to permute the position of the outmost boxes since they can not be used as center boxes in subroutine `scatter` and are taken out of the loop there by simply starting the loop variable at for example `-mgridxy+1`. This simple means of skipping the outmost boxes would not work if the index for `mgridxy` would also have been permuted:

```
      do 20 k=-mgridxy,mgridxy
c Find entry numbers, don't permutate outmost boxes
         i=int((1.-2.*ran1(iseed))*(mgridxy-2))
         j=int((1.-2.*ran1(iseed))*(mgridxy-2))
```

```
c Exchange
           exch=permx(i)
           permx(i)=permx(j)
           permx(j)=exch
c Find entry numbers, don't permutate outmost boxes
           i=int((1.-2.*ran1(iseed))*(mgridxy-2))
           j=int((1.-2.*ran1(iseed))*(mgridxy-2))
c Exchange
           exch=permy(i)
           permy(i)=permy(j)
           permy(j)=exch
20      continue
```

Since permz has another size than permx and permy it is permuted separately:

```
        do 30 k=-mgridz,mgridz
c Find entry numbers, don't permutate outmost boxes
           i=int((1.-2.*ran1(iseed))*(mgridz-2))
           j=int((1.-2.*ran1(iseed))*(mgridz-2))
c Exchange
           exch=permz(i)
           permz(i)=permz(j)
           permz(j)=exch
30      continue
```

Finally deltax, deltay and deltaz are permuted:

```
        i=1+int(3.*ran1(iseed))
        j=1+int(3.*ran1(iseed))
        exch=deltx(i)
        deltx(i)=deltx(j)
        deltx(j)=exch
c
        i=1+int(3.*ran1(iseed))
        j=1+int(3.*ran1(iseed))
        exch=delty(i)
        delty(i)=delty(j)
        delty(j)=exch
c
        i=1+int(3.*ran1(iseed))
        j=1+int(3.*ran1(iseed))
        exch=deltz(i)
        deltz(i)=deltz(j)
        deltz(j)=exch
c
        return
        end
```

B.14 DISTRI

CTEQ Collaboration

Last revision of the file: 10/7/93

These subroutines calculate the parton distribution function ctq2pd as they will be used by init.

```
      FUNCTION Ctq2Pd (Iset, Iparton, X, Q, Irt)

C     Version 2 CTEQ distribution function in a parametrized form.

C By: J. Botts, H.L. Lai, J.G. Morfin, J.F. Owens, J. Qiu, W.K. Tung & H.Weerts

C  To avoid the proliferation of parton distribution functions, we recommend  that
C  these distributions should replace Version 1 CTEQ distributions for all general
C  usage.

C     (No data tables are needed.)

C The returned function value is the PROBABILITY density for a given FLAVOR.

C  !! A companion function (next module), which this one depends on,
C  !!        Ctq2df (Iset, Iparton, X, Q, Irt)
C  !! gives the VALENCE and SEA MOMENTUM FRACTION distributions.

C  !! Also included are two functions PrCtq2 & Wlamd2 which return relevant
C  !! QCD parameters associated with these parton distributions sets.(See  below)

C  \\  A parallel (independent) program Ctq2Pds (not included in this file)
C  ||  in Subroutine form is also available.
C  ||  It returns ALL the parton flavors at once in an array form.
C  //  See details in that separate file if you are interested.

C     Since this is an initial distribution, and there may be updates, it is
C     useful for the authors to maintain a record of the distribution list.
C     Please do not freely distribute this program package; instead, refer any
C     interested colleagues to direct their request for a copy to:
C     Botts@hades.ifh.de or Lai@cteq11.pa.msu.edu

C If you have detailed questions concerning these distributions, direct inquires
C to Botts, Lai (see above) or Wu-Ki Tung (Tung@msupa.pa.msu.edu).

C   This function returns the CTEQ parton distributions f^Iset_Iprtn/proton
C     where Iset (= 1, 2, ..., 5) is the set label;


C       Name convention for CTEQ distributions:  CTEQnSx  where
C           n : version number                      (currently n = 1)
C           S : factorization scheme label: = [M D L] for [MS-bar DIS LO]

c                resp.
C           x : special characteristics, if any

C                (e.g. S(F) for singular (flat) gluon, L for "LEP lambda value")

C   Iprtn  is the parton label (6, 5, 4, 3, 2, 1, 0, -1, ......, -6)
C                          for (t, b, c, s, d, u, g, u_bar, ..., t_bar)
```

```
C X, Q are the usual x, Q; Irt is a return error code (not implemented yet).

C --> Iset = 1, 2, 3, 4, 5 correspond to the following CTEQ global fits:
C     cteq2M, cteq2MS, cteq2MF, cteq2ML, cteq2L  respectively.

C --> QCD parameters for parton distribution set Iset can be obtained inside
C         the user's program by:
C     Dum = Prctq2

C    >        (Iset, Iord, Ischeme, MxFlv,
C    >         Alam4, Alam5, Alam6, Amas4, Amas5, Amas6,
C    >         Xmin, Qini, Qmax, ExpNor)
C     where all but the first argument are output parameters.
C     They should be self-explanatory -- see details under ENTRY Prctq2.

C  Since the QCD Lambda value for the various sets are needed more often than
C  the other parameters in most applications, a special function

C     Wlamd2 (Iset, Iorder, Neff)                    is provided
C  which returns the lambda value for Neff = 4,5,6 effective flavors as well as
C  the order these values pertain to.

C     The range of (x, Q) used in this round of global analysis is, approxi-
C     mately,  0.01 < x < 0.75 ; and 4 GeV^2 < Q^2 < 400 GeV^2 for fixed target
C     experiments and 0.0001 < x < 0.01 from preliminary data of HERA.

C    The range of (x, Q) used in the reparametrization of the QCD evolved
C    parton distributions is 10E-5 < x < 1 ; 1.6 GeV < Q < 1 TeV.  The

C    functional form of this parametrization is:

C      A0 * x^A1 * (1-x)^A2 * (1 + A3 * x^A4) * [log(1+1/x)]^A5

C    with the A'coefficients being smooth functions of Q.  For heavy quarks,
C    a threshold factor is applied to A0 which simulates the proper Q-dependence
C    of the QCD evolution in that region.

C    Since this function is positive definite and smooth, it provides sensible
C    extrapolations of the parton distributions if they are called beyond
C    the original range in an application. There is no artificial boundaries
C    or sharp cutoff's.
```

Appendix C

Index

Bibliography

[Aic91] J. Aichelin, Phys. Rep. **202**, 233 (1991); H. Feldmeier, Nucl. Phys. **A515**, 147 (1990); A. Ono *et al.*, Prog. of Theoret. Phys. **87**, 1185 (1992); H. Sorge, H. Stöcker, and W. Greiner, Nucl. Phys. **A498**, 567c (1989); H. Sorge, H. Stöcker, and W. Greiner, Ann. Phys. **192**, 266 (1989).

[Aic96] J. Aichelin, Phys. Rev. **C33**, 537 (1986); H. Stöcker, and W. Greiner, Phys. Rep. **137**. 277 (1986); U. Mosel, Annu. Rev. Nucl. Part. Sci. **41**, 29 (1991); W. Bauer *et al.*, Annu. Rev. Nucl. Part. Sci. **42**, 77 (1992); P. Schuck *et al.*, Prog. Part. Nucl. Phys. **22**, 181 (1989); Y. Pang, T. Schlagel, and S. H. Kahana, Nucl. Phys. **A544**, 435c (1992); D. E. Kahana, D. Keane, Y. Pang, T. Schlagel, and S. Wang, Phys. Rev. Lett. **74**, 4404 (1995).

[Ale95] F. J. Alexander, A. L. Garcia, and B. J. Alder, Phys. Rev. Lett. **74**, 5212 (1995)

[Alm95] T. Alm *et al.*, Nucl. Phys. **A587**, 815 (1995)

[Bad89] H. Badovsky, Eur. J. Mech., **B**/Fluids **8**, 41 (1989)

[Bau84] W. Bauer, D. R. Dean, U. Mosel, and U. Post, in *Procedings of the 7th High Energy Heavy Ion Study* (Report GSI-85-10, 1984), 701; W. Bauer, D. R. Dean, U. Mosel, an U. Post, Phys. Lett **150B**, 53 (1985); W. Bauer, U. Post, D. R. Dean, and U. Mosel, Nucl. Phys. **A452**, 699 (1986); W. Bauer, Phys. Rev. **C38**, 1297 (1988); W. Bauer and A. Botvina, Phys. Rev. **C52**, R1760 (1996).

[Bau86] W. Bauer, G. F. Bertsch, W. Cassing, and U. Mosel, Phys. Rev. **C34**, 2127 (1986); W. Bauer, Nucl. Phys. **A471**, 604 (1987); W. Bauer, G. F. Bertsch, and H. Schulz, Phys. Rev. Lett. **69**, 1888 (1992).

[Ber88] G. F. Bertsch and S. Das Gupta, Phys. Rep. **160, No. 4**, 189 (1988)

[Bot93] Version 2 CTEQ distribution function in a parametrized form, J. Botts, H. L. Lai, J. G. Morfin, J. F. Owens, J. Qiu, W.-K. Tung and H. Weerts, CTEQ Collaboration.

[Bro96] Hartree-Fock Code DENS, B. A. Brown, priv. communication.

[Cam85] X. Campi and J. Debois, in *Proceedings of the 23rd Bormeo Conference* (Ricerca Scientifica et Educatione Permanente, Milano, 1985), 497; T. S. Biro, J. Knoll, and J. Richert, Nucl. Phys. **A459**, 692 (1986); X. Campi,

J. Phys. **A19**, L917 (1986); J. Nemeth, M. Barranco, J. Debois, and C. Ngô, Z. Phys. **A325**, 347 (1986); J. Debois, Nucl. Phys. **A466**, 724 (1987); C. Cerruti, J. Debois, R. Boisgard, C. Ngô, J. Natowitz, and J. Nemeth, Nucl. Phys. **A476**, 74 (1988); S. Das Gupta, C. Gale, and K. Haglin, Phys. Lett. **302B**, 372 (1993).

[Com77] B. L. Combridge, J. Kripfganz and J. Ranft, Phys. Lett. **70B**, No. 2, 234 (1977)

[Con79] A. Coniglio, H. E. Stanley, and W. Klein, Phys. Rev. Lett. **42**, 518 (1979); D. W. Hermann and D. Stauffer, Z. Phys. **B44**, 339 (1981).

[Cug81] J. Cugnon, T. Mizutani, J. Vandermeulen, Nucl. Phys. **A352**, 505 (1981)

[Cze86] L. P. Czernai, and J. I. Kapusta, Phys. Rep. **131**, 223 (1986); S. Das Gupta, and G. D. Westfall, Physics Today **46**(5), 34 (1993); R. Stock, Phys. Rep. **135**, 259 (1986); H. Gutbrod *et al*, Rep. Prog. Phys. **52**, 1267 (1989).

[Dan91] P. Danielewicz and G. F. Bertsch, Nucl. Phys. **A533**, 712, appendix (1991)

[Dan95] P. Danielewicz, Phys. Rev. **C51**, 716 (1995)

[Dan96] P. Danielewicz, and S. Pratt, Phys. Rev. **C53**, 249 (1996)

[Das93] S. Das Gupta and G. D. Westfall, G. D., 1993, Physics Today **46**(5), 34 (1993)

[Eic84] E. Eichten, I. Hinchliffe, K. Lane and C. Quigg, Rev. Mod. Phys., Vol. **56**, No. 4, 579 (1984)

[Fie89] R. D. Field, *Frontiers in Physics: Applications of Perturbative QCD*, Addison-Wesley 1989

[Gei92.1] K. Geiger and B. Müller, Nucl. Phys. **A544**, 467c (1992)

[Gei92.2] K. Geiger and B. Müller, Nucl. Phys. **B369**, 600 (1992)

[Gei93] K. Geiger, Phys. Rev. Lett. **71**, No. 19, 3075 (1993)

[Gei94.1] K. Geiger, Phys. Rev. **C49**, 3234 (1994)

[Gei94.2] K. Geiger, private communication

[Gus88] H. A. Gustafson *et al.* (Plastic Ball Collaboration), Mod. Phys. Lett. A **3**, 1323 (1988)

[Kal93] U. Kalmbach, T. Vetter, T. S. Biró and U. Mosel, Nucl. Phys. **A563**, 584 (1993)

[Kod84] T. Kodama, S. B. Duarte, K. C. Chung, R. Donangelo and R. A. M. S. Nazareth, Phys. Rev. **C29**, 2146 (1984)

[Kod94] T. Kodama, private communication

[Kor93] G. Kortemeyer, J. Murray, S. Pratt, K. Haglin and W. Bauer, NSCL/Cyclotron Laboratory Annual Report 1993, 63, 65

[Kun96] G. J. Kunde et al., Phys. Rev. Lett **77** (1996), 2897.

[Lan92] A. Lang, H. Badovsky, W. Cassing, U. Mosel, H.-G. Reusch, and K. Weber, J. Comp. Phys. **106**, 391 (1993)

[Lan93] A. Lang, H. Badovsky, W. Cassing, U. Mosel, Hans-Georg Reusch and Klaus Weber, J. Comp. Phys. **106**, 391 (1993)

[Mcl82] L. McLerran, *Quark Matter and Heavy Ion Collisions; Bielefeld Workshop 1982*, World Scientific, 63

[Mor97] K. Morawetz et al., Universität Rostock, in preparation.

[Mur93] J. Murray, G. Kortemeyer, S. Pratt, K. Haglin and W. Bauer, NSCL/Cyclotron Laboratory Annual Report 1993, 61

[Par95] M. D. Partlan *et al.* (EOS Collaboration), Phys. Rev. Lett. **75**, 2100 (1995)

[Pha92] L. Phair et al., Phys. Lett. **285B**, 10 (1992); L. Phair, W. Bauer and C. K. Gelbke, Phys. Lett. **314B**, 271 (1993).

[Res77] P. M. Résibois, and M. De Leener, *Classical Kinetic Theory of Fluids*, p. 156, John Wiley & Sons, New York, 1977

[Sor89.1] H. Sorge, H. Stöcker and W. Greiner, Nucl. Phys. **A498**, 567c (1989)

[Sor89.2] H. Sorge, H. Stöcker and W. Greiner, Ann. Phys. **192**, 266 (1989)

[Sta79] D. Stauffer, Phys. Rep. **54C**, 1 (1979); J. W. Essam, Rep. Prog. Phys. **43**, 883 (1980).

[Tsa93] M. B. Tsang et al., Phys. Rev. Lett **71**, 1502 (1993)

[Wel89] G. Welke, R. Malflied, C. Grégoire, M. Prakash and E. Suraud, Phys. Rev. **C40**, 2611 (1989)

[Wes86] G. D. Westfall *et al.*, Phys. Rev. Lett. **71**, 1986 (1993); D. Kakow, G. Welke, and W. Bauer, Phys. Rev. **C48**, 1982 (1993).