11-19-2008

Do R_{AA} and R_{CP} Quantify Nuclear Medium Effects?

Robert Adrian Zaballa

DO R_{AA} AND R_{CP} QUANTIFY NUCLEAR MEDIUM EFFECTS?

by

ROBERT A. ZABALLA

Under the Direction of Xiaochun He

ABSTRACT

With the use of an incoherent binary nucleon-nucleon collision model of heavy ion collisions for simulating particle production, it is demonstrated that the nuclear modification factors, R_{AA} and R_{CP}, are less than unity for hard scattering in the absence of any nuclear modification effects. The nuclear modification factor R_{dAu} is also shown to approach or exceed unity only if p_T broadening is taken into account. With a simple phenomenological parameter, the mean nucleon energy loss fraction, this model yields particle distributions that are comparable to those of experiment. The nuclear geometry is described by the Glauber model, and particle production is simulated by the PYTHIA event generator.

INDEX WORDS: Heavy ion collisions, Nuclear modification factors, Incoherent binary nucleon-nucleon collisions, Hard scattering, p_T broadening, Glauber model

DO R_{AA} AND R_{CP} QUANTIFY NUCLEAR MEDIUM EFFECTS?

by

ROBERT A. ZABALLA

A Dissertation Submitted in Partial Fullfilment of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2008

DO R_{AA} AND R_{CP} QUANTIFY NUCLEAR MEDIUM EFFECTS?

by

ROBERT A. ZABALLA

Committee Chair: Dr. Xiaochun He

Committee: Dr. Nikolaus Dietz
Dr. Steven Manson
Dr. Richard Miller
Dr. A. G. Unil Perera
Dr. Murad Sarsour

Electronic Version Approved:

Office of Graduate Studies
College of Arts and Sciences
Georgia State University
December 2008

To my parents, Maria and Angel Zaballa

ACKNOWLEDGEMENTS

I would first like to thank my parents and siblings for always being there for me, as well as my friends, who have all been supportive. They have always provided me with great inspiration to pursue my goals, no matter how difficult the tasks involved. They were always understanding of my dedication to a deeper understanding of the laws of nature, and my devotion to my work. My parents always encouraged and supported my interests in science, from buying me my first telescope, to providing me with books that I would read in my spare time.

My brother, Alex, would help me with my computers from time to time, and he helped me to better understand them. My sister, Angela, would cook and bake cookies for me to cheer me up whenever I felt down. Whenever I needed to take a break, I could also count on them to go and do something fun. Furthermore, I give thanks to my grandparents for their wisdom and for coming to the United States from Cuba. My maternal grandfather, Roberto Rodriguez, in particular, nurtured my interest in science when I would read from his collection of science magazines. My uncles also inspired me to pursue science, as we would have discussions on astronomy and physics.

My graduate studies in high energy nuclear physics would not have been possible without Dr. Xiaochun He, from whom I have learned so much, and who has been a great mentor and friend. He has provided me with great insight in conducting research. Under his guidance, I am greatful to have taken part in such a rich area of study, and to have broadened my experience with computational techniques and research in general. My graduate years

have been an enlightening experience that could not have proceeded as they have without Xiaochun's advisement. I appreciate all of his suggestions and efforts that have helped me to excel.

I also give thanks to Dr. David Ritz Finkelstein and Dr. John Wood for their guidance during my undergraduate studies at the Georgia Institute of Technology. From my early years as a teenager, David helped me to understand the foundations of physics and to start my research career. My initial research interests were in grand unification and quantum gravity. Then, my interest in nuclear physics began when I took part in research led by John, and his insights helped me to advance my abilities as a researcher. I thank Dr. Cheuk-Yin Wong of the University of Tennessee and Oak Ridge National Laboratory for valuable suggestions during the course of my graduate studies, and all members of the Nuclear Physics Group at Georgia State University for their support as well. Dr. Jun Ying was very helpful in assisting me with computer related matters.

Before my college years, my thirst for knowledge was recognized by several of my teachers in high school. I appreciate the support of Coach Strickland, Coach Wall, Mrs. Lucking, Mrs. Hunsucker, and Mrs. Hayes at Etowah High School. I excelled in my high school studies, exempting algebra and calculus, taking advanced mathematics courses at Kennesaw State University, and studying relativity, quantum mechanics, and quantum field theory in my spare time. These early years of development paved the way for my future studies.

Thank you all.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AA	Nucleus-Nucleus
Au	Gold
Au + Au	Gold + Gold
ch	**Ch**arged Particle
coll	Binary **Coll**isions
cms	**C**enter of **M**ass **S**ystem
CP	**C**entral to **P**eripheral
Cu	Copper
Cu + Cu	Copper + Copper
d	**D**euteron
dA	Deuteron-Nucleus
dAu, d + Au	Deuteron + Gold
ev, evt	**Ev**ent
fm	**F**emto**m**eter
GB	**G**iga**b**yte
GeV	**G**iga**e**lectron **V**olt
inel	**Inel**astic
h	**H**adron
HIJING	**H**eavy **I**on **J**et **IN**teraction **G**enerator
lab	**Lab**oratory

LEXUS	**L**inear **EX**trapolation of **U**ltrarelativistic **S**cattering
mb	**M**illi**b**arn
MB	**M**ega**b**yte
MeV	**M**ega**e**lectron **V**olt
NN	**N**ucleon-**N**ucleon
part	**Part**icipant
PHENIX	**P**ioneering **H**igh **E**nergy **N**uclear **I**nteraction E**X**periment
PHOBOS	An experiment at RHIC named after a moon of Mars
pp	**P**roton-**P**roton
PYTHIA	An event generator named after a priestess in Greek mythology
QCD	**Q**uantum **C**hromo**d**ynamics
QED	**Q**uantum **E**lectro**d**ynamics
QGP	**Q**uark-**G**luon **P**lasma
RHIC	**R**elativistic **H**eavy **I**on **C**ollider
UA1	An experiment at the European Organization for Nuclear Research

CHAPTER 1

INTRODUCTION

Understanding the behavior of nuclear matter under different temperature and density conditions is a subject of intense experimental and theoretical work in nuclear physics. Recent results at the Relativistic Heavy Ion Collider (RHIC) suggest the formation of an ultradense form of matter[1]. Cold nuclear matter effects need to be understood before firm conclusions can be drawn on the nature of the hot nuclear medium that has been produced at RHIC. For example, although the quark-gluon plasma (QGP) is not expected to be formed in proton-proton (pp), proton-nucleus (pA), or deuteron-nucleus (dA) collisions, these collisions result in effects which must be ascertained separately from those that may be associated with the QGP[2]. In particular, the energy loss of nucleons in these collisions must be considered.

Many particle production models have been formulated in the past. For example, the Heavy Ion Jet INteraction Generator (HIJING) is an elegant model often used to study phenomenological aspects of nuclear collisions on the parton level, but it introduces

additional processes, such as jet quenching and nuclear shadowing, for consistency with experimental data[2]. In HIJING, additional, more complicated mechanisms, such as baryon junction loops, may also be introduced at the expense of simplicity[3]. Also, the Linear EXtrapolation of Ultrarelativistic Scattering (LEXUS) model incorporates a Glauber-like approach (see Chapter 4) with no free parameters, all information coming from simple parameterizations of nucleon-nucleon collision data[4]. This model, although elegant and similar in many respects to the model presented here, is complicated in that it requires numerical solutions for nucleon distribution functions that cannot be solved in closed form.

While it is difficult to single out a particular theoretical model of particle production from available data, it is very instructive to formulate a simple binary nucleon collision model to estimate contributions to particle production from multiple nucleon-nucleon (NN) collisions, with nucleon energy loss taken into account. Furthermore, whereas many processes require a description in terms of partons (quarks and gluons), there are aspects of nuclear collisions that may be described in terms of nucleons[5].

In models of parton energy loss, it is often assumed that since it is unlikely for a parton to undergo more than one hard collision, the particle yields from these collisions should scale with the number of binary collisions in the absence of nuclear effects[2]. Therefore, it is assumed that for the case of independent NN collisions, and in the absence of any nuclear effects at high p_T ($p_T > 2$ GeV/c), $R_{AA} = 1$, where R_{AA} is the nuclear modification factor that compares the particle yield from nucleus-nucleus (AA) collisions to that of pp collisions[6, 7, 8, 9]. This however, could only be the case if each NN collision were of the same energy as the pp collision energy in the reference spectrum appearing in the denomi-

nator of R_{AA}. Furthermore, as it is assumed that nucleons undergo multiple collisions which contribute to particle production, these nucleons must lose energy to the particle production process, and therefore subsequent binary collisions should occur with lower energies. Then, hard scatterings of partons in these subsequent binary collisions should also occur at lower energies than the initial hard scatterings. The nuclear modification factor would then be less than unity.

A simple phenomenological model is presented that emphasizes the basic features of multiple nucleon collisions, utilizing a Glauber Monte Carlo description. Distributions of produced particles, obtained from the model, are comparable to those of experiment. It is demonstrated that R_{AA} and R_{CP} are less than unity for hard scattering in the absence of any nuclear modification effects. R_{dAu} is also shown to approach or exceed unity only if p_T broadening is taken into account. It is then concluded that the nuclear modification factors do not quantify nuclear medium effects.

In this dissertation, basic concepts of high energy nuclear physics are introduced that are necessary for an understanding of the presented study. In Chapter 2, the physics of quarks, gluons, and the quark-gluon plasma is introduced. Nucleon-nucleon collisions, in particular, proton-proton collisions, are introduced in Chapter 3. A brief discussion of the Glauber model of nucleus-nucleus collisions is then given in Chapter 4, followed by an introduction to nuclear effects in Chapter 5. After these introductory chapters, the formulation of the incoherent binary collision model and the simulation procedure are presented in Chapter 6. The results of the model are then presented and discussed in Chapter 7, and the summary and conclusions are given in Chapter 8. The appendices

provide further background material. First, Appendix A introduces relativistic kinematics, variables, and cross sections. Then, Appendix B provides an introduction to the PYTHIA event generator. Finally, Appendix C contains the codes of the C++ program files of runpythia and glauber_mc, as well as header and implementation files used by the programs.

CHAPTER 2

QUARKS, GLUONS, AND THE QUARK-GLUON PLASMA

The fundamental theory of the strong nuclear interaction is known as *quantum chromodynamics* (QCD). In this theory, the fundamental particles that interact are quarks and gluons. Quarks are fermions of spin 1/2, and gluons are bosons of spin 1, in a manner analogous to quantum electrodynamics (QED) where electrons are fermions (spin 1/2) and photons are bosons (spin 1). Whereas in QED the internal degree of freedom of interaction is electric charge, in QCD, the analogous degree of freedom is known as *color*. The term *chromodynamics* is derived from this color concept[10].

2.1 Quarks, Gluons, and Hadrons

Quarks are known to exist in six different types, or *flavors*, each corresponding to a particular mass and electric charge. These six flavors are, in order of increasing mass: *up*,

down, *strange*, *charm*, *bottom*, and *top*, and are denoted by u, d, s, c, b, and t, respectively. There also exist six *antiflavors* of antiquarks with equal mass, and equal but opposite electric charge to the corresponding flavors. These antiflavors are, in correspondence to the six quark flavors, *antiup*, *antidown*, *antistrange*, *anticharm*, *antibottom*, and *antitop*, and are denoted by $\bar{u}$, $\bar{d}$, $\bar{s}$, $\bar{c}$, $\bar{b}$, and $\bar{t}$, respectively. The masses and electric charges of the six quark flavors are given in Table 2.1, where the masses are given in MeV/c^2 for the three lighter quarks (u, d, and s), and GeV/c^2 for the three heavier quarks (c, b, and t). The electric charge is in units of, e, the absolute value of the electron charge[11].

Table 2.1: The six flavors of quarks.

Flavor	Mass	Electric Charge (e)
u	1.5 - 3.0 MeV/c^2	2/3
d	3 - 7 MeV/c^2	-1/3
s	95 $\pm$ 25 MeV/c^2	-1/3
c	1.25 $\pm$ 0.09 GeV/c^2	2/3
b	4.20 - 4.70 GeV/c^2	-1/3
t	172 - 174 GeV/c^2	2/3

Quarks interact with each other, or with antiquarks, through the exchange of gluons, which are massless, and gluons may also interact with each other. Unlike electric charge, color, carried by quarks, takes on three distinct values: *red*, *blue*, and *green* (not to be confused with the usual notion of color). In addition, there are three anticolors carried by antiquarks: *antired*, *antiblue*, and *antigreen*. Whereas a quark can carry any one color, a gluon carries one color and one anticolor. As a result, there are eight different gluons, and they may interact with each other in addition to interacting with quarks[12].

The color interaction is short-ranged, confined to within particles called *hadrons*,

consisting of quarks and antiquarks. All hadrons are found to be color neutral, which must be the case if the color interaction is to be confined within them. There are two classes of hadrons: *baryons* and *mesons*. Baryons consist of three quarks. There also exist antibaryons, such as antiprotons and antineutrons, consisting of three antiquarks. Mesons consist of a quark and an antiquark. Examples of baryons are shown in Table 2.2, and mesons in Table 2.3, along with their masses, electric charge, and valence quark structures[11, 13]. In baryons, color neutrality is realized by each of the three quarks having one of the three distinct colors, and in mesons, this neutrality is realized by the quark having a particular color, and the antiquark having the corresponding anticolor.

Table 2.2: Examples of baryons.

Baryon	Mass (MeV/c^2)	Electric Charge (e)	Valence Quarks
p	938.3	1	uud
n	939.6	0	udd
Σ^+	1189.4	1	uus
Σ^0	1192.6	0	uds
Σ^-	1197.4	-1	dds
Λ	1115.7	0	uds

Table 2.3: Examples of mesons.

Meson	Mass (MeV/c^2)	Electric Charge (e)	Valence Quarks
π^+, π^-	139.6	+1, -1	$u\bar{d}$, $d\bar{u}$
π^0	135.0	0	$(u\bar{u} - d\bar{d})/\sqrt{2}$
K^+, K^-	493.7	+1, -1	$u\bar{s}$, $s\bar{u}$
K^0, $\bar{K}^0$	497.7	0	$d\bar{s}$, $s\bar{d}$
ϕ	1019.4	0	$s\bar{s}$
J/ψ	3096.9	0	$c\bar{c}$
Υ	9460.3	0	$b\bar{b}$

Experimentally, it is found that quarks are tightly bound within hadrons through a mechanism known as *confinement*, which is characteristic of QCD. However, in physical processes with higher momentum transfers, the QCD coupling constant is found to decrease and approach zero[13]. This property of the coupling is known as *asymptotic freedom* and suggests that at higher momentum scales, or smaller distance scales, the strong interaction diminishes, and on these scales, quarks and gluons are not as tightly confined as they are at lower momentum scales (larger distance scales). The higher strength of the interaction at larger distance scales prevents color sources from being separated so that they are confined to within smaller distance scales. Confinement is then a result of this behavior of the QCD coupling.

Quarks and gluons are referred to collectively as *partons*. The first indication of the partonic structure of hadrons came from electron-proton collision experiments[14]. Large transfers of energy and momentum suggest that the electric charge of the proton is localised on a few scattering centers. In hadron-hadron collisions, partons from each hadron collide, resulting in a process known as *fragmentation*. The incoming partons polarize the surrounding vacuum, creating more partons. For example, as a quark-antiquark pair is pulled apart, as illustrated in Fig. 2.1, newly created partons combine into hadrons due to confinement. In electron-hadron collisions the electron interacts with a parton in the hadron. In electron-positron collisions, the electron and positron annihilate to form a virtual photon which may then transform into a quark-antiquark pair. The partonic structure of a hadron is often characterized by a parton structure function, which gives the probability of finding a parton with a certain fraction of its parent hadron's total momentum[13].

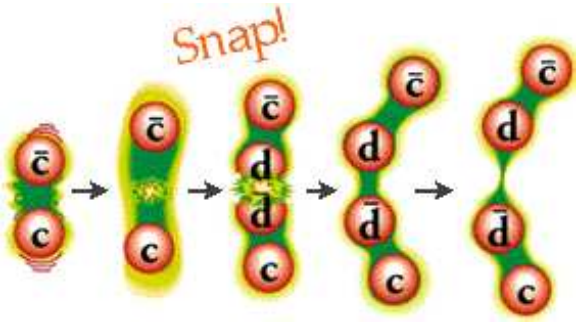

Figure 2.1: (color online) The fragmentation of a quark-antiquark pair[12].

2.2 The Quark-Gluon Plasma

Previously it was mentioned that at higher momentum scales, the QCD coupling diminishes via asymptotic freedom. As this coupling is reduced at high momentum transfers, so too is the degree of confinement that binds quarks together within hadrons, and it should be possible for quarks and gluons to deconfine for a sufficiently weak coupling. Such a deconfinement, expected to occur at sufficiently high energy density and temperature, would result in the formation of the *quark-gluon plasma* (QGP)[13]. A phase transition would then occur from a state of matter consisting of hadrons, to a deconfined state where quarks and gluons may freely propagate on larger distance scales (scales larger than the size of a hadron).

A hadron may be thought of as a *bag* in which quarks and gluons reside. In this bag, the valence quarks interact through the exchange of gluons, and virtual quark - antiquark pairs also form. If the density of hadrons in a region of space is increased, the hadrons may coalesce into an extended region in which quarks and gluons may propagate and are deconfined. An increase in hadronic density can result either by compressing hadronic

matter, in particular, by raising the baryon density, or by heating it to a sufficiently high temperature that hadron production occurs, thus raising the hadronic density. It is thought that in the early universe, approximately 10^{-5} - 10^{-4} seconds after the Big Bang[15], matter in the universe existed in this high temperature state of deconfined quarks and gluons.

Through a rise in temperature of hadronic matter above a critical temperature, $T_c \sim 150$ - 200 MeV, the QGP is expected to form. The expected energy density of the deconfined matter is ~ 1 GeV fm^{-3} or higher[15]. This critical temperature, which is significantly higher than the temperature of the center of the Sun, $T \approx 1.3$ keV, is typical of hadronic interactions and can be achieved in accelerators. In the formation of the QGP at high baryon density, baryons will overlap at a critical baryon density and dissociate into degenerate quark matter. This condition of high baryon density is expected to occur in certain astrophysical situations, such as in the core of superdense stars, like neutron stars. If the density at the center of a neutron star reaches 5 - 10 ρ_{nm}, where $\rho_{nm} \approx 0.16$ fm^{-3} is the baryon density of normal nuclear matter, then degenerate quark matter may form[15].

The phase diagram for nuclear matter is shown in Fig. 2.2, with temperature plotted versus baryon density. Normal nuclear matter is found in the region of lower temperature and baryon density. Just beyond the region of normal nuclear matter, the hadronic gas phase is found with higher temperature or baryon density. Beyond the hadronic gas phase is the QGP, at higher temperatures and baryon densities. The early universe is indicated as having existed in the QGP phase at high temperature and with a baryon density comparable to that of normal nuclear matter. At very high baryon density and low temperature, there is a phase known as the *color superconductor* in which there is expected to be

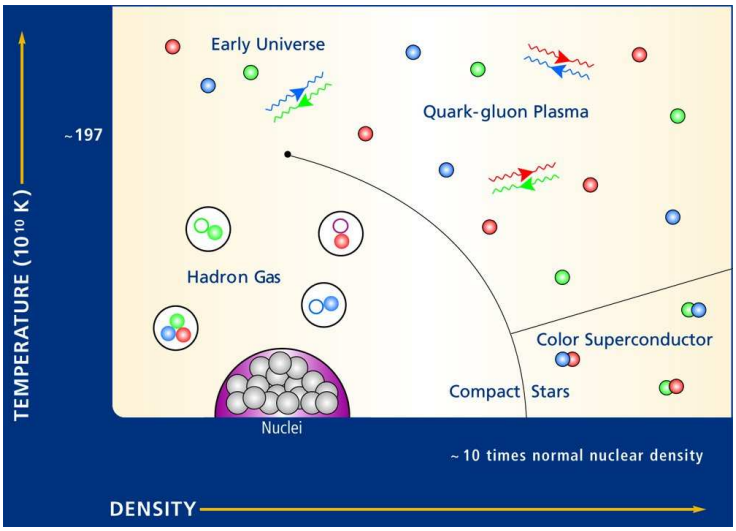

Figure 2.2: (color online) Phase diagram of QCD[16].

a long-range gluonic attraction between quarks, leading to the formation of quark Cooper pairs, analogous to the formation of Cooper pairs of electrons in superconductors[15].

In the laboratory, it may be possible to form the QGP by colliding heavy nuclei, such as gold (Au), at relativistic energies. Such experiments are being performed at the *Relativistic Heavy Ion Collider* (RHIC) at Brookhaven National Laboratory. In high energy collisions, the incoming nuclei are Lorentz-contracted as they travel at velocities close to the speed of light (roughly 99.95%), as illustrated in the first panel of Fig. 2.3. For collision energies of a few GeV per nucleon, the heavy ions stop each other, resulting in a high baryon density. At significantly higher energies, on the order of 100 GeV per nucleon, the heavy ions pass through each other, leaving a region of high energy density and temperature between the ions as they recede from each other[13], as illustrated in the second and third panels. This region of high energy density is also very low in baryon density. It is in this region of high energy density that the QGP may briefly form. As the region of high energy density

expands, it cools and the produced matter transforms, after approximately 10^{-23} seconds, into hadrons and other particles that may be detected, as illustrated in the fourth panel. It is these detected particles that provide insight into the nature of heavy ion collisions.

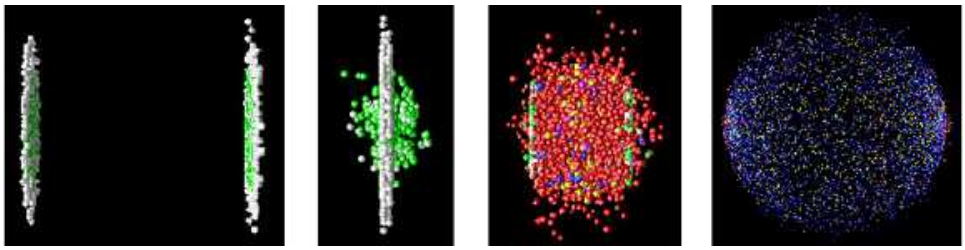

Figure 2.3: (color online) The phases of a heavy ion collision[17].

CHAPTER 3

NUCLEON-NUCLEON COLLISIONS

As discussed previously, it is expected that the QGP may be formed by compressing baryonic matter or by an increase in the energy density of a hadronic system. Each of these conditions may be realized in heavy-ion collisions. A heavy-ion is an atomic nucleus stripped of its atomic electrons. The nucleus consists of protons and neutrons, referred to collectively as *nucleons*. In heavy-ion collisions, the nucleons from one nucleus collide with those of the other. The trends in particle production in heavy ion collisions can then be compared to particle production in simpler systems, such as pp collisions.

3.1 Elastic and Inelastic pp Collisions

The total cross section for pp collisions is shown in Fig. 3.1 and is expressed in millibarns (mb), where 1 mb = 0.1 fm^2 = 10^{-31} m^2. Contributions to the total cross

section are from *elastic* collisions, in which the nucleons do not lose any energy, and *inelastic* collisions, in which the nucleons lose energy, and the lost energy goes into the production of particles. The inelastic cross section is given by the difference of the total and elastic cross sections. The tendency of nucleons to lose energy to particle production increases with collision energy. For collision energies above roughly 4 GeV, the nucleon-nucleon inelastic cross section is approximately 40 mb, which is much higher than the elastic cross section. The elastic cross section decreases from approximately 10 mb at $\sqrt{s} \approx 4$ GeV to about 7 mb at around 100 GeV.

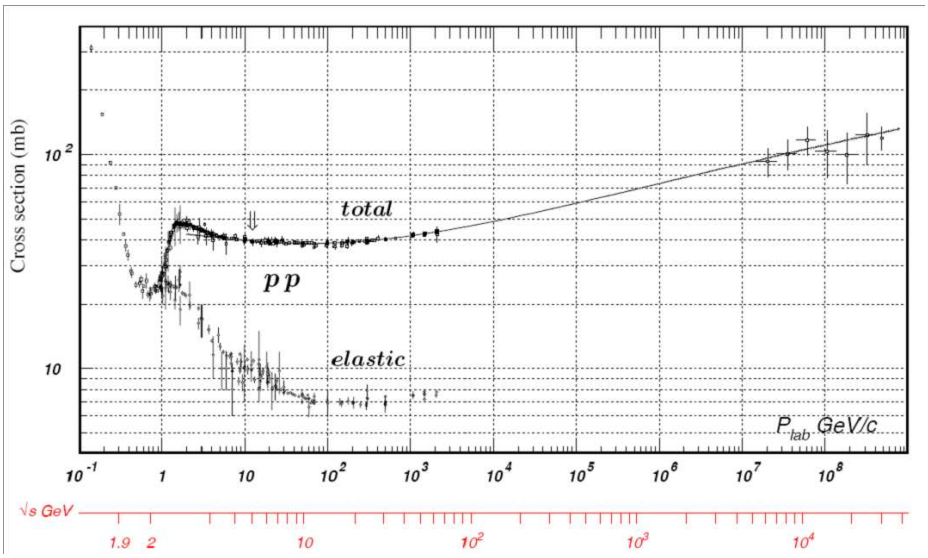

Figure 3.1: (color online) Total and elastic cross sections of pp collisions in mb, shown as a function of lab momentum and center of mass energy[18].

When considering inelastic collisions, there is *diffractive scattering* in which one nucleon acts as a region of absorption, and the interference of scattering amplitudes from different impact parameters of the colliding nucleons results in a diffraction pattern in the forward and backward directions[10]. Upon undergoing diffractive scattering the nucleons may only be slightly excited, resulting in little energy loss, and the production of very

few particles. In nondiffractive inelastic collisions the nucleons lose a significant amount of energy to particle production. Attention will be given to nondiffractive inelastic collisions, and they will be referred to simply as inelastic collisions.

3.2 Multiplicity Distributions

As the collision energy of the nucleons is increased, the number of particles produced in nucleon-nucleon collisions, or *multiplicity*, increases as $\sim \ln \sqrt{s}$, where $\sqrt{s}$ is the center of mass energy of the collision[15]. Shown in Fig. 3.2 is the mean charged particle multiplicity as a function of collision energy for e^+e^-, $p(\bar{p})$-p, and $e^{\pm}p$ collisions. Roughly

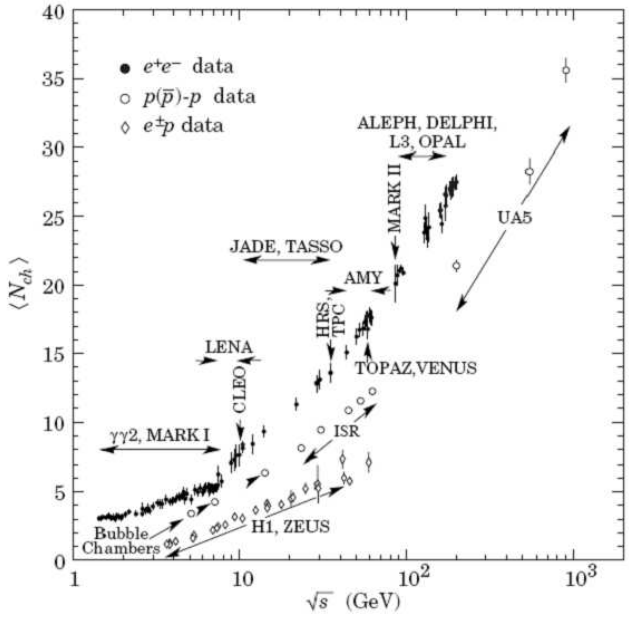

Figure 3.2: Charged particle multiplicity vs. center of mass energy for e^+e^-, $p(\bar{p})$-p, and $e^{\pm}p$ collisions. Experiments from which the data points were obtained are indicated[18].

80 - 90% of produced particles are pions[10]. The inclusive charged particle pseudorapidity density distribution, $\frac{1}{N_{ev}}\frac{dN}{d\eta}$ (see Section A.3), for $p\bar{p}$ collisions is shown in Fig. 3.3 for

various collision energies. The height of the distribution increases with collision energy. The dip in the pseudorapidity distribution at mid-rapidity becomes more pronounced as the collision energy is increased.

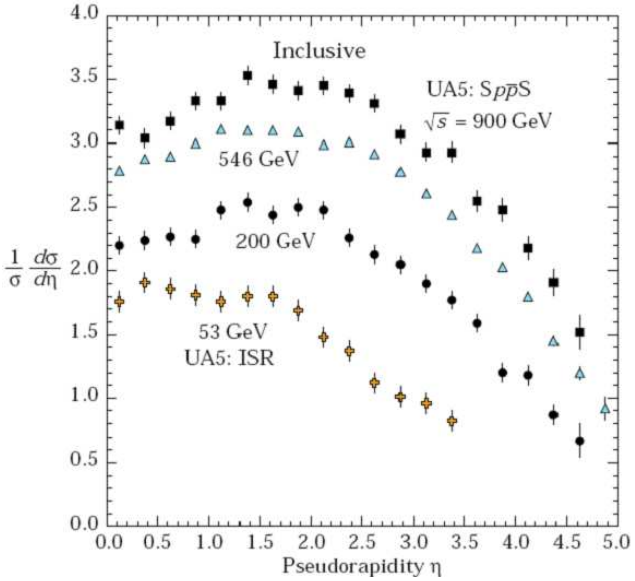

Figure 3.3: (color online) Charged particle pseudorapidity density distributions for p$\bar{p}$ collisions at various collision energies[18].

3.3 Transverse Momentum Distributions

Produced particles are also characterized by their momentum distributions. The longitudinal momentum distribution can be obtained from the pseudorapidity distribution. Transverse momentum (p_T) distributions are shown in Fig. 3.4 for charged particles from pp collisions at various collision energies. These distributions take on an approximate exponential shape, and the distribution decreases more rapidly at small p_T than at large p_T. Particles with transverse momenta less than 1 - 2 GeV/c tend to be produced in *soft* partonic interactions, which are described by non-perturbative QCD (low momentum

transfer), and particles above this range tend to originate from *hard* partonic interactions, which are described by perturbative QCD (high momentum transfer). Soft interactions occur with a higher probability than hard interactions. However, soft interactions are more difficult to understand due to their nonperturbative nature. Strictly speaking, there is no clear cut separation between the contributions of soft and hard collisions, but contributions from hard collisions become more dominant as p_T increases.

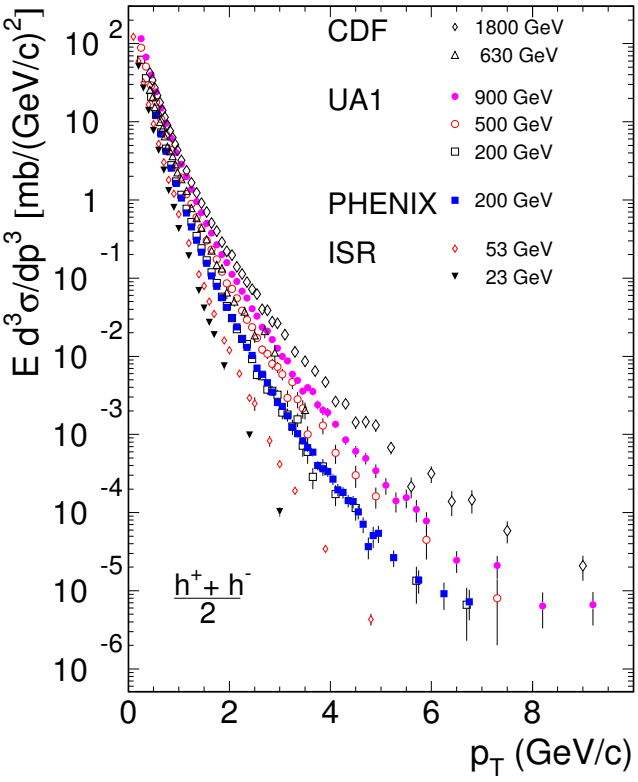

Figure 3.4: (color online) p_T distributions for charged particles from pp collisions for various collision energies at mid-rapidity[19].

CHAPTER 4

THE GLAUBER MODEL

High energy collisions of nuclei yield a significantly greater multiplicity of produced particles than elementary proton-proton collisions. In a collision of nuclei, thousands of particles can be produced. Since nuclei consist of nucleons, it is important to understand how the constituent nucleons contribute to particle production. The most successful model that describes the collision of nuclei in terms of their constituent nucleons is the *Glauber model*. In this model, the nucleons of the colliding nuclei undergo multiple collisions. After an incident nucleon suffers a collision, it may be considered to reemerge from the collision, and continue to collide with other nucleons.

4.1 Nucleus-Nucleus Collisions

A nucleus-nucleus collision is characterized by its impact parameter b, the distance between the centers of the nuclei, perpendicular to the beam axis. Nucleons may be considered to be aligned along rows that are parallel to the beam axis. Each row is then

characterized by its own impact parameter. A nucleon in a particular row will suffer multiple collisions if it is aligned with a row of nucleons in the incoming nucleus as shown in Fig. 4.1. Designating one nucleus as the target nucleus, and the other as the projectile nucleus, projectile nucleons are assumed to collide only with target nucleons. The projectile nucleus is taken to travel along the positive z-direction, and the target nucleus is either taken to be at rest, relative to the laboratory, or to travel in the opposite direction.

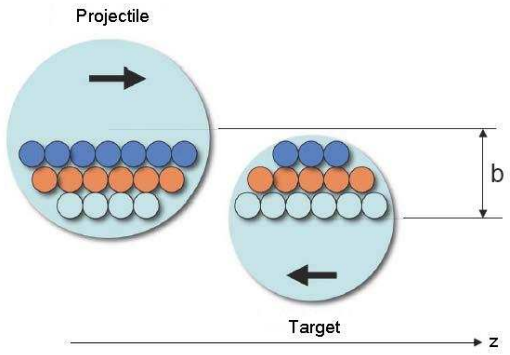

Figure 4.1: (color online) Glauber geometry of a nucleus-nucleus collision. Rows of nucleons from the projectile nucleus collide with oncoming rows of the target nucleus. The centers of the nuclei are separated by the impact parameter b.

In a nucleus-nucleus collision, there will be a region of overlap between the nuclei in which nucleon-nucleon collisions occur, and outside of which nucleon-nucleon collisions do not occur, as illustrated in Fig. 4.2. The size of this *overlap region* is dependent on the impact parameter. For zero impact parameter, the collision is completely head-on, and the overlap region is its maximum size. If the impact parameter is greater than the sum of the radii of the two nuclei, there will be no overlap region. Nucleus-nucleus collisions for which the impact parameter is relatively small are called *central collisions*, and those collisions with large impact parameters are called *peripheral collisions*.

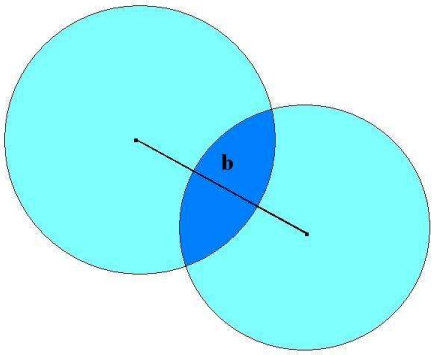

Figure 4.2: (color online) The overlap region of two colliding nuclei, indicated by the dark blue region, for impact parameter b, as viewed in the plane perpendicular to the beam axis.

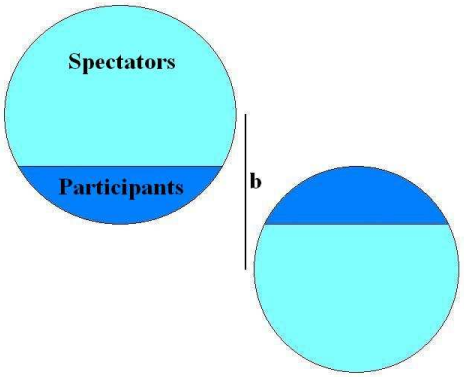

Figure 4.3: (color online) The regions of two colliding nuclei, indicated by the dark blue regions, for impact parameter b, where the participants are found. These regions intersect to form the overlap region.

Nucleons that suffer at least one nucleon-nucleon collision are called *participants*, and those that do not suffer any collisions are called *spectators*, as shown in Fig 4.3. The spectator nucleons will continue to move along the beam axis. A nucleon-nucleon collision is also referred to as a *binary collision*. At small impact parameters there are higher numbers of participants and binary collisions than at larger impact parameters for which there may be no participants or binary collisions. The mean numbers of binary collisions $\langle N_{coll} \rangle$ and participants $\langle N_{part} \rangle$, versus impact parameter, are shown in Fig. 4.4 for Au + Au and Cu

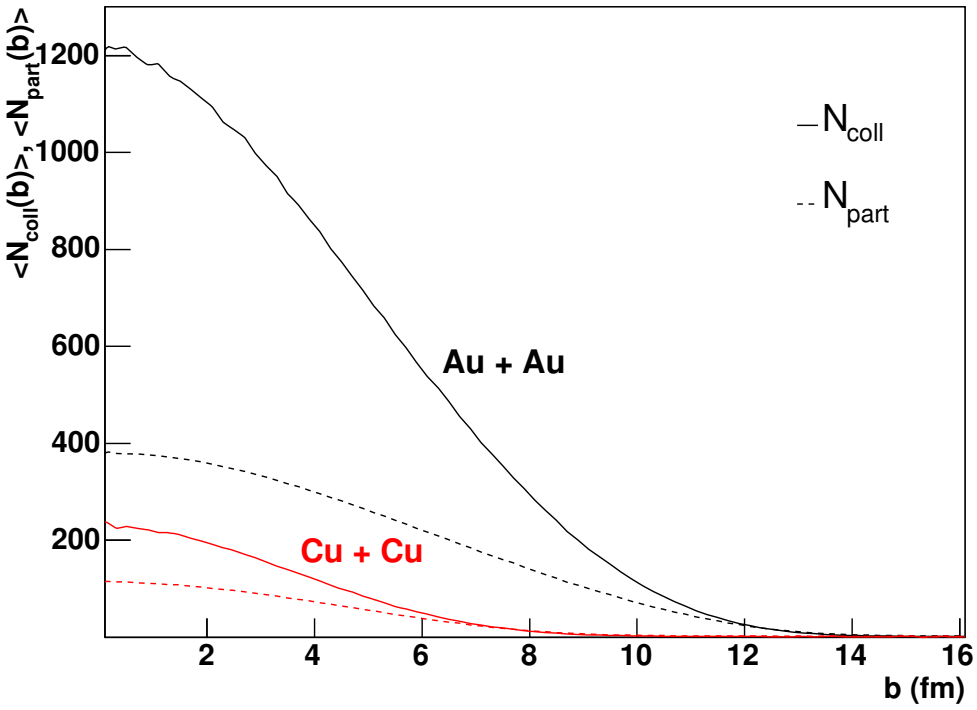

Figure 4.4: (color online) $\langle N_{coll} \rangle$ and $\langle N_{part} \rangle$ vs. b.

+ Cu collisions at $\sqrt{s_{NN}} = 200$ GeV. The impact parameter, number of participants and number of binary collisions cannot be measured directly in experiments, but are estimated in the Glauber model.

4.2 Experimental Input

Any Glauber model approach requires the input of experimental data. Two important experimental inputs are the nuclear density, which specifies the distribution of nucleons within a nucleus, and the inelastic nucleon-nucleon cross section, for characterizing binary collisions.

4.2.1 Nuclear Density

The nucleon density distribution is given by[20]

$$\rho(r) = \rho_0 \frac{1 + w(r/R)^2}{1 + e^{\left(\frac{r-R}{a}\right)}}, \tag{4.1}$$

where ρ_0 is the nucleon density at the center of the nucleus, r is the distance of a nucleon from the center of the nucleus, R is the nuclear radius, a is the "skin depth", also known as the "diffuseness parameter," and w is a measure of the deviation of the nuclear shape from a sphere. The nuclear radius may be expressed in terms of the atomic number A as $R = r_0 A^{1/3}$, where r_0 is the nucleon radius[15]. For ^{197}Au, $R = 6.38$ fm, $a = 0.54$ fm, $w = 0$, and for ^{63}Cu, $R = 4.21$ fm, $a = 0.60$ fm, $w = 0$. These are nuclei that are used in RHIC experiments.

Another important nucleus in experiments at RHIC is the deuteron, consisting of a proton and neutron. The distance between the proton and neutron is obtained from the *Hulthén* wave function

$$\phi(r_{pn}) = \frac{1}{\sqrt{2\pi}} \frac{\sqrt{ab(a+b)}}{b-a} \frac{e^{-ar_{pn}} - e^{-br_{pn}}}{r_{pn}}, \tag{4.2}$$

where r_{pn} is the distance between the proton and neutron, and the parameters a and b have

the values 0.228 fm^{-1} and 1.18 fm^{-1}, respectively[20].

4.2.2 Inelastic Nucleon-Nucleon Cross Section

The inelastic nucleon-nucleon cross section, as a function of collision energy, is another experimental input in Glauber calculations. The nucleon inelastic cross section increases with collision energy from about 32 mb at $\sqrt{s_{NN}} = 20$ GeV, to around 42 mb at $\sqrt{s_{NN}} = 200$ GeV, where $\sqrt{s_{NN}}$ is the nucleon-nucleon center of mass energy (see Section 3.1). Furthermore, in Glauber model calculations, the nucleon may be treated as an object with a finite extent. If it is assumed that for two incident nucleons, there is always a nucleon-nucleon collision as long as they overlap (aligned along the same row as in Fig. 4.1), then the nucleons are referred to as *black disks*. If it is assumed that there is only a probability of a nucleon-nucleon collision as the two nucleons approach each other and overlap, then the nucleons are referred to as *gray disks*. Depending on the model, the probability for a collision of gray disks can be dependent on the degree of overlap of the two nucleons.

4.3 The Inelastic Nucleus-Nucleus Cross Section and Centrality Classes

The numbers of binary collisions and participants cannot be directly measured experimentally. However, their mean values can be estimated for classes of nuclear collision events in a Glauber model calculation. The inelastic nucleus-nucleus differential cross section is shown in Fig. 4.5 as a function of the impact parameter b for Au + Au, Cu + Cu, and d + Au collisions. An inelastic nucleus-nucleus collision is one for which there is

at least one binary collision. The linear part of the differential cross section is given by $d\sigma/db = 2\pi b$. This linear form would continue as b increases if all events were counted, that is, inelastic collisions and those events for which the impact parameter is so large that the nuclei do not collide. For low to moderate values of the impact parameter, all or most events are inelastic, corresponding to the linear part of the cross section, but for large impact parameters, fewer events are inelastic, so the inelastic differential cross section drops off and approaches 0 as $b \to \infty$.

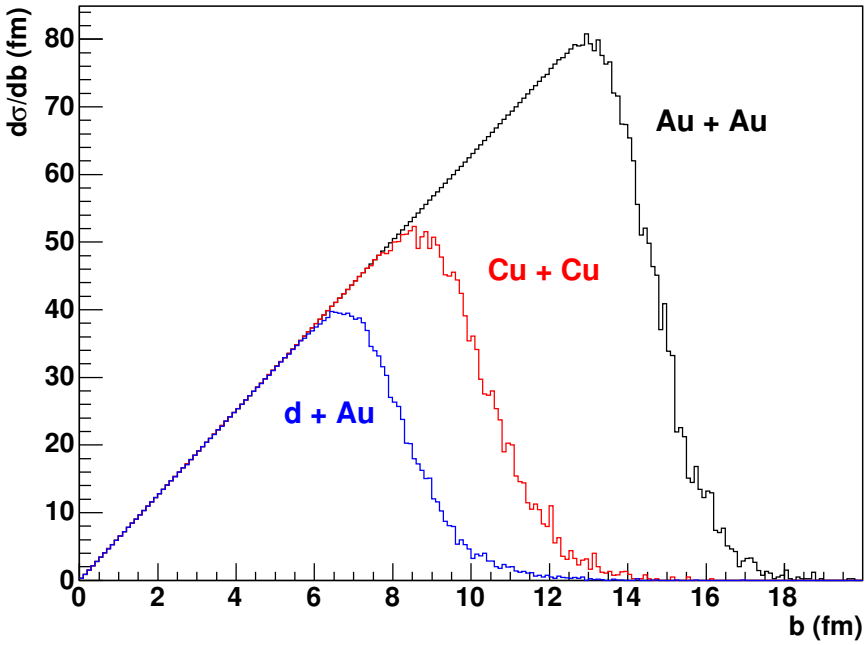

Figure 4.5: (color online) The inelastic nucleus-nucleus differential cross section vs. b, for Au + Au, Cu + Cu, and d + Au collisions. Each centrality class corresponds to a range of percentages of the total cross section.

The total areas under the $d\sigma/db$ curves are equal to the total inelastic cross sections of these collision systems. For Au + Au, this total cross section is approximately 6,850 mb, for Cu + Cu it is about 3,420 mb, and for d + Au it is about 2,180 mb. Fractions of the

inelastic cross sections may be expressed as percentages of the total inelastic cross sections for each collision system. In this way, *centrality classes* may be defined. Each centrality class corresponds to an area under the $d\sigma/db$ curve that runs from one value of the impact parameter to another. For example, the 0 - 10% centrality class for Au + Au is the portion of the area under the $d\sigma/db$ curve that is bounded by $b = 0$ fm and $b = 4.6$ fm. This portion then corresponds to 0% - 10% of the total inelastic cross section. Similarly there is the 10% - 20% centrality class, which for Au + Au, is bounded by $b = 4.6$ fm and $b = 6.6$ fm. The mean numbers of binary collisions and participants may then be obtained for each centrality class, as well as the number of particles produced from collisions within each class. The mean numbers of binary collisions and participants for Au + Au at $\sqrt{s_{NN}} = 200$ GeV, from PHOBOS[21] calculations, are shown in Table 4.1. The mean numbers of binary collisions and participants for d + Au at $\sqrt{s_{NN}} = 200$ GeV, from PHENIX[22] calculations, are shown in Table 4.2.

Table 4.1: $\langle N_{coll} \rangle$ and $\langle N_{part} \rangle$ for Au + Au collisions at $\sqrt{s_{NN}} = 200$ GeV, from PHOBOS[21].

Centrality	$\langle N_{coll} \rangle$	$\langle N_{part} \rangle$
0% - 6%	1040	344
6% - 15%	762	276
15% - 25%	483	200
25% - 35%	286	138
35% - 45%	164	93
45% - 50%	99	65

Table 4.2: $\langle N_{coll} \rangle$ and $\langle N_{part} \rangle$ for d + Au collisions at $\sqrt{s_{NN}} = 200$ GeV, from PHENIX[22].

Centrality	$\langle N_{coll} \rangle$	$\langle N_{part} \rangle$
0% - 20%	15.4	15.0
20% - 40%	10.6	10.4
40% - 60%	7.0	7.0
60% - 88%	3.1	3.2

CHAPTER 5

NUCLEAR EFFECTS IN HEAVY ION COLLISIONS

In the study of high energy nucleus-nucleus collisions, it is necessary to understand the properties of the nuclear medium under various conditions. It is expected that, in the initial stages of a central relativistic collision of heavy nuclei, such as Au, normal nuclear matter should dissociate into a deconfined phase of quarks and gluons. However, in order to determine whether the QGP has been created in these collisions, it is also important to understand those phenomena in nucleus-nucleus collisions not associated with the QGP. These *cold nuclear matter* phenomena may be studied in smaller collision systems, such as proton-proton (pp), proton-nucleus (pA), and deuteron-nucleus (dA) collisions, where energy densities are not expected to be sufficient for the formation of a deconfined medium. In this chapter, particle spectra of heavy ion collisions are first introduced, followed by a discussion of nuclear effects.

5.1 Particle Spectra of Heavy Ion Collisions

The pseudorapidity distributions for Au + Au collisions at $\sqrt{s_{NN}} = 19.6$, 130, and 200 GeV are shown in Fig. 5.1 for many centralities from PHOBOS[21]. Central collisions result in higher yields than peripheral collisions. The distributions become wider and higher as the collision energy is increased.

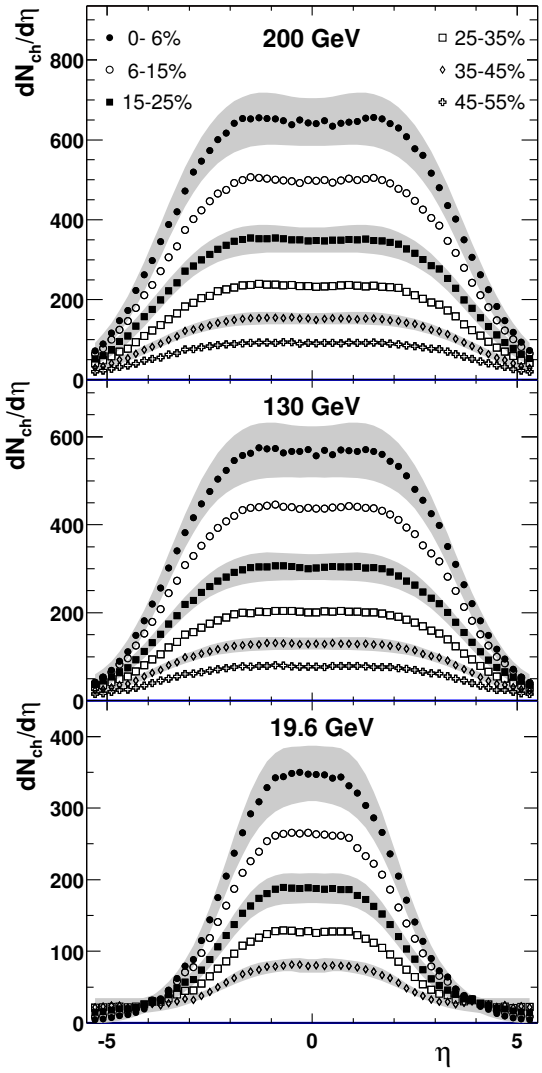

Figure 5.1: Charged particle pseudorapidity distributions for Au + Au collisions at $\sqrt{s_{NN}} =$ 19.6, 130, and 200 GeV for several centralities. The experimental data are from PHOBOS[21].

The charged particle pseudorapidity distributions for d + Au at $\sqrt{s_{NN}} = 200$ GeV are shown in Fig. 5.2 for five centralities and for minimum bias (all nucleus-nucleus collisions). The pseudorapidity is measured relative to the nucleon-nucleon center of mass frame. Negative pseudorapidity corresponds to the direction of motion of the gold nucleus. The mean pseudorapidity is negative for central collisions, indicating the net longitudinal momentum of the participant nucleons, and for peripheral collisions, the mean pseudorapidity is zero, as their are an equal number of participants traveling in opposite directions.

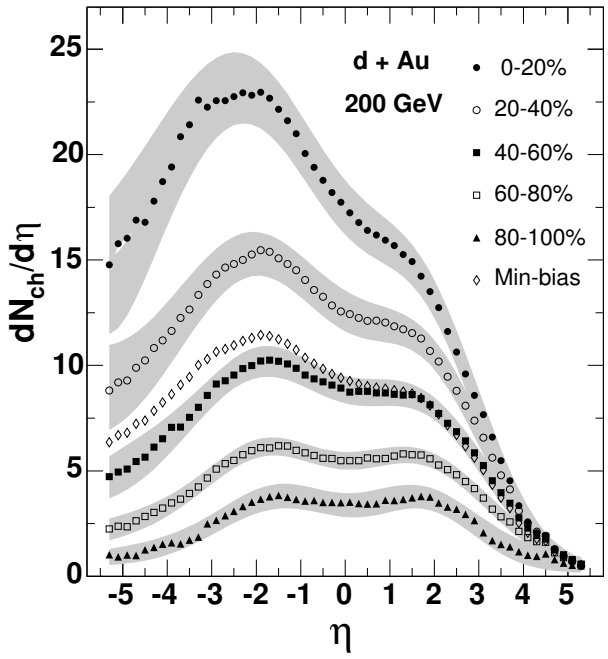

Figure 5.2: Charged particle pseudorapidity distributions for d + Au collisions at $\sqrt{s_{NN}} =$ 200 GeV, for minimum bias and several centralities. The experimental data are from PHOBOS[23].

The p_T spectra for charged particles are shown in Fig. 5.3, for Au + Au collisions at $\sqrt{s_{NN}} = 200$ GeV for minimum bias and various centralities at mid-rapidity. At high p_T the

spectra have power-law tails, and the shapes of the spectra are more concave for peripheral collisions than for central collisions. Particles in the region $p_T < 2$ GeV/c tend to originate from soft parton interactions, and particles beyond this region from hard interactions[24]. More will be said about soft and hard interactions in connection with nuclear medium effects in Section 5.2.

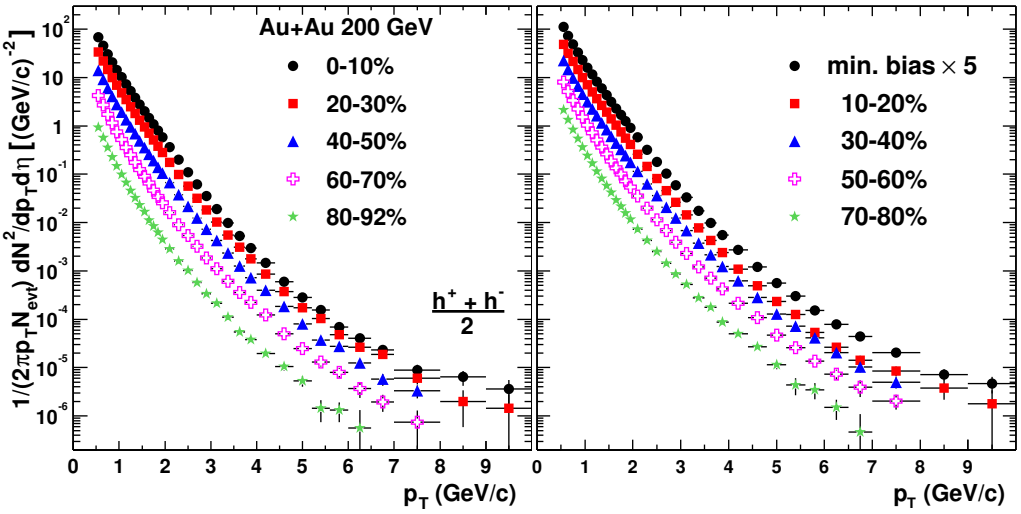

Figure 5.3: (color online) Charged particle p_T distributions for Au + Au collisions at $\sqrt{s_{NN}} = 200$ GeV for several centralities and $|\eta| < 0.18$. The experimental data are from PHENIX[24].

5.2 Nuclear Effects

During an AA collision, as nucleons collide, their constituent partons interact. Various effects associated with these parton interactions may be classified as *initial state* or *final state* effects. Initial state effects occur before the formation of any dense medium, and also occur in systems for which no dense medium is expected to form. Final state effects are associated with the produced medium, as the medium itself influences parton processes.

To understand nuclear effects, it is useful to model nucleus-nucleus collisions as a sum of *incoherent* binary collisions, that is, collisions that are independent of one another in the particle production process. In this simple case, there are no nuclear medium effects, so any deviations in observed yields from the idealized particle yields of the incoherent case should indicate nuclear effects. To quantify these effects, the nuclear modification factors, R_{AA} and R_{CP}, are defined as[19]

$$\begin{aligned} R_{AA} &= \frac{1}{\langle N_{coll}\rangle}\left[\frac{1}{2\pi p_T N_{ev}}\frac{d^2N}{dp_T d\eta}\right]_{AA} \\ &\times \left[\frac{1}{2\pi p_T N_{ev}}\frac{d^2N}{dp_T d\eta}\right]_{pp}^{-1}, \end{aligned} \tag{5.1}$$

and

$$\begin{aligned} R_{CP} &= \left[\frac{1}{\langle N_{coll}\rangle}\frac{1}{2\pi p_T N_{ev}}\frac{d^2N}{dp_T d\eta}\right]_{C} \\ &\times \left[\frac{1}{\langle N_{coll}\rangle}\frac{1}{2\pi p_T N_{ev}}\frac{d^2N}{dp_T d\eta}\right]_{P}^{-1}, \end{aligned} \tag{5.2}$$

where $\langle N_{coll}\rangle$ is the mean number of binary collisions for a given centrality class, N_{ev} is the number of AA collisions (for a given centrality class) or pp collisions, as specified by the square brackets $[\,]_{AA}$ or $[\,]_{pp}$, and N is the number of produced particles, which may be, for example, total charged particles ($\pi^{\pm}$, $K^{\pm}$, p, and $\bar{p}$) or neutral pions (π^0). The subscripts C and P, in the definition of R_{CP}, refer respectively, to the most central and peripheral centrality classes.

The nuclear modification factor R_{AA}, shown in Fig. 5.4, compares the yield from AA collisions to that of elementary pp collisions, and R_{CP}, shown in Fig. 5.5, compares the yields of central AA collisions to those of peripheral collisions. Assuming multiple binary collisions, it is highly unlikely that any parton suffers more than one hard collision, so one

pair of partons may collide per hard collision[2, 15], and the particle yield from hard collisions should scale with the number of binary collisions in the incoherent case. Thus, deviations of R_{AA} from unity at high p_T, corresponding to hard parton collisions, should indicate nuclear medium effects.

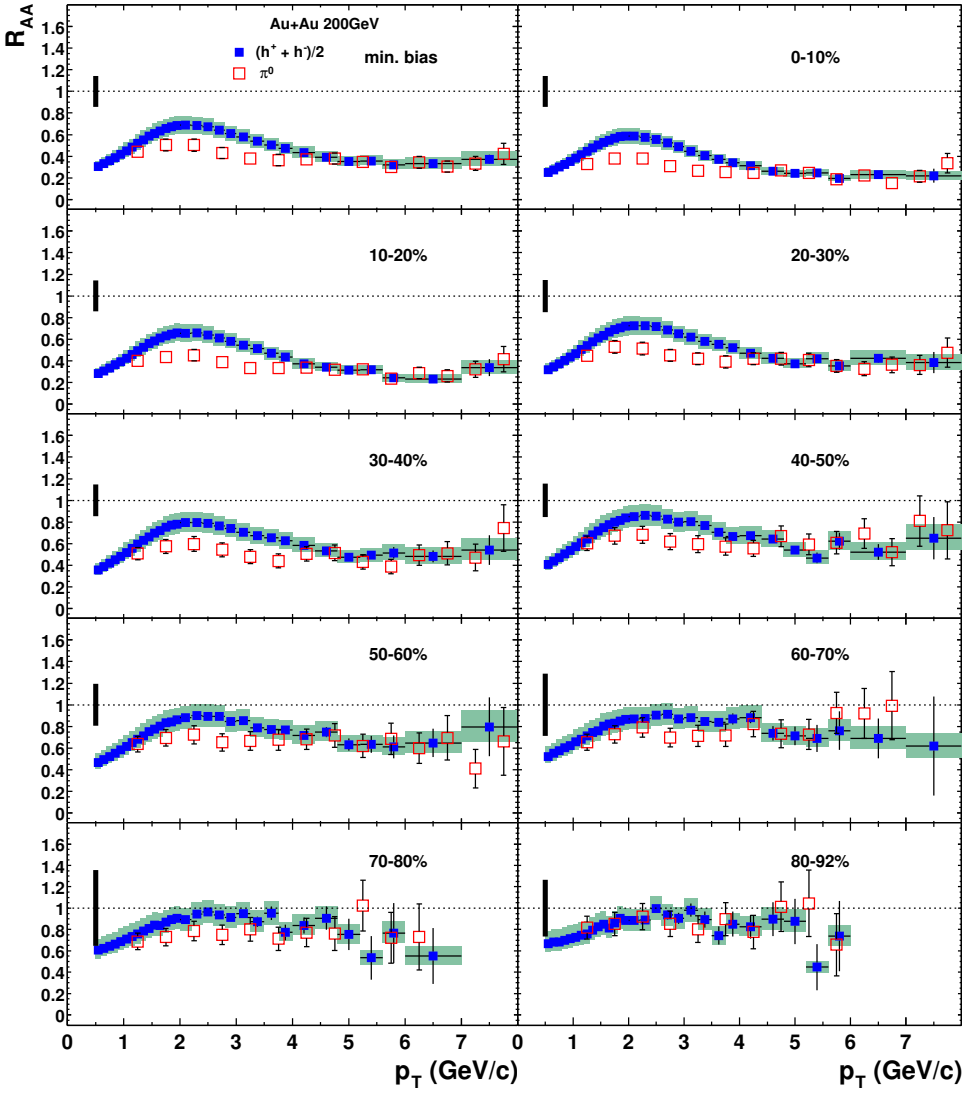

Figure 5.4: (color online) Charged particle and π^0 R_{AA} distributions for Au + Au collisions at $\sqrt{s_{NN}}$ = 200 GeV, for several centralities and $|\eta| < 0.35$. The experimental data are from PHENIX[19].

From Fig. 5.4, R_{AA} is seen to increase with p_T for $p_T < 2$ GeV/c, and reach a maximum at $p_T \approx 2$ GeV/c. Furthermore, the maximum R_{AA} for charged particles is higher than that of the π^0 R_{AA}. The higher maximum of the charged particle R_{AA} is due to an enhancement in the baryon spectra for $2 < p_T < 4.5$ GeV/c. This baryon enhancement is not entirely understood, though there are several models that have been proposed to explain it[24]. This enhancement is more pronounced for central collisions than for peripheral collisions, resulting in the peak at $p_T \approx 2$ GeV/c in R_{CP}, as shown in Fig. 5.5.

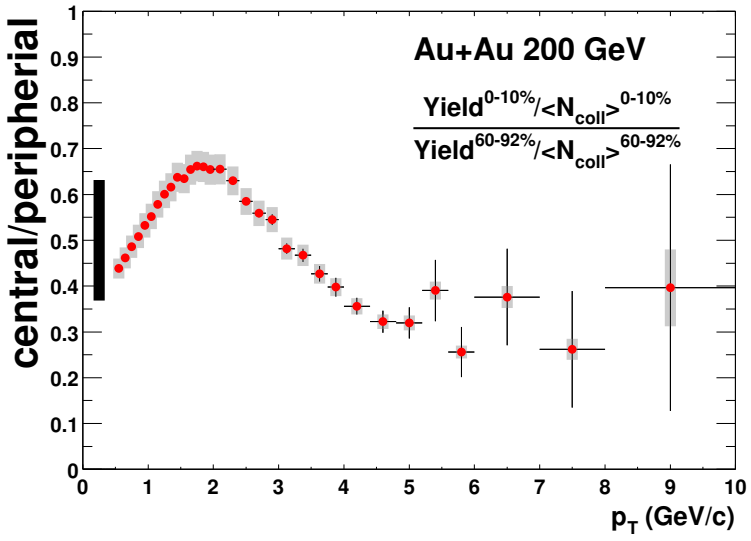

Figure 5.5: (color online) Charged particle R_{CP} distribution for Au + Au collisions at $\sqrt{s_{NN}} = 200$ GeV and $|\eta| < 0.35$. The experimental data are from PHENIX[19].

As partons collide, they tend to lose energy, resulting in the production of hadrons and other particles that can be detected. Various mechanisms regulate particle production and depend on the collision system. One observation, from the above figures for the nuclear modification factors, is that, when compared to yields from pp collisions, there is a *suppres-*

sion of hadron yields, at high transverse momentum, in central Au + Au collisions. The hadron yield from Au + Au collisions is about a factor of 5 less than that of pp collisions when scaled by the number of binary collisions[25]. Also, the yield from central collisions is suppressed relative to that of peripheral collisions. This suppression has been predicted to be a consequence of parton energy loss in the produced medium[19], though the precise mechanisms that give rise to it are not fully understood.

In lighter systems, such as d + Au, the suppression of hadron yields is not observed, and no dense medium is expected to be produced. This observation rules out any initial state effects that could give rise to a suppression in particle yields, and final state energy loss in d + Au collisions is expected to be small[25]. By examining particle yields in lighter systems and comparing them to those of heavier systems, it is possible to distinguish between initial and final state effects.

A comparison of the nuclear modification factors of d + Au collisions to those of Au + Au collisions at $\sqrt{s_{NN}}$ = 200 GeV, is shown in Fig. 5.6. Whereas suppression is observed for the central Au + Au yield, it is not observed for d + Au, and in fact, R_{dAu} is seen to exceed unity for $p_T > 2$ GeV/c. The d + Au yield is enhanced relative to pp collisions. This enhancement, known as p_T *broadening*, or the *Cronin effect*[25, 26], is an enhancement of p_T spectra in the p_T range 2 - 5 GeV/c, and is generally attributed to initial state multiple soft parton scattering before a hard scattering. This scattering results in an increase in the transverse momentum of partons and a broadening of the parton p_T distribution[22]. The enhancement in baryon yields mentioned earlier may also be due to a centrality and particle species dependent p_T broadening effect[24, 27].

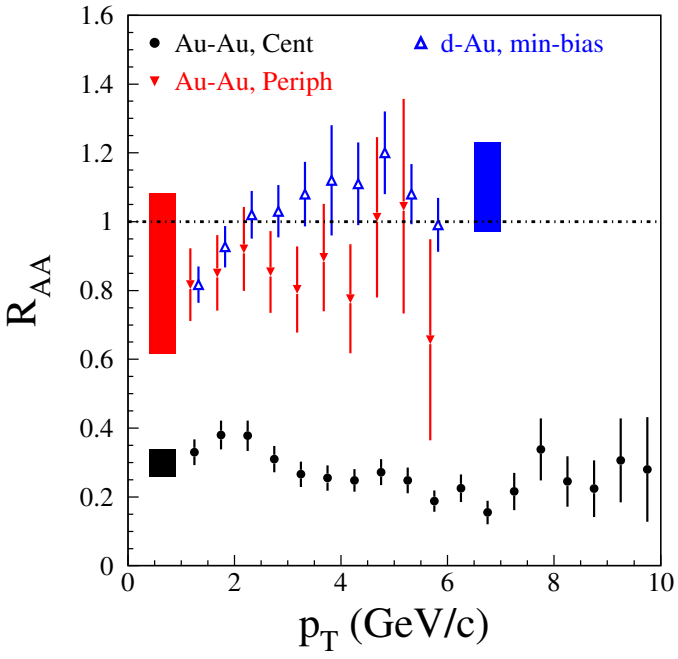

Figure 5.6: (color online) π^0 R_{AA} distributions for central and peripheral Au + Au collisions at $\sqrt{s_{NN}} = 200$ GeV, and R_{dAu} for minimum bias d + Au collisions at 200 GeV and $|\eta| < 0.35$. The experimental data are from PHENIX[19].

CHAPTER 6

THE INCOHERENT BINARY COLLISION MODEL

A model of particle production from incoherent binary nucleon-nucleon collisions is presented. The nuclear geometry of the model is presented first, followed by the nucleon collision kinematics, the settings and implementation used in PYTHIA (see Appendix B), and a description of the Glauber Monte Carlo simulation program. The simulation procedure is then discussed.

6.1 Collision Geometry

In the simulation program, the impact parameter b of a nucleus-nucleus collision is randomly assigned from the distribution $d\sigma/db$, where σ is the inelastic nucleus-nucleus cross section[20] (see Section 4.3). Nucleons are randomly distributed in a nucleus according

to the nucleon density, which is parameterized by the Woods-Saxon distribution,

$$\rho(r) = \rho_0 \frac{1}{1 + e^{\frac{r-R}{a}}}, \tag{6.1}$$

where ρ_0 is the nucleon density in the center of the nucleus, R is the nuclear radius, and a is the diffuseness parameter, or "skin depth". For ^{197}Au, $R = 6.38$ fm and $a = 0.54$ fm, and for ^{63}Cu, $R = 4.21$ fm and $a = 0.60$ fm. The radial distance of a nucleon from the center of its parent nucleus is then randomly assigned from the distribution $4\pi r^2 \rho(r)$. For d + Au collisions, the proton-neutron distance is determined by the Hulthén wave function, as in Subsection 4.2.1.

For a given impact parameter, the nuclear overlap geometry is given by the Glauber model description, as illustrated in Fig. 4.1. For each nucleon, either from the projectile or target nucleus, it will only interact with nucleons aligned along its direction of travel (assumed to be parallel to the direction of motion of its parent nucleus). It is assumed that nucleon motion is one-dimensional so that projectile nucleons always move (before and after a binary collision) along the positive z-axis, and target nucleons always move along the negative z-axis. Projectile nucleons experience collisions only from target nucleons and the produced particles do not experience any secondary interactions. Furthermore, a binary collision occurs if the distance d between the two nucleons, transverse to the beam axis, satisfies the inequality,

$$d \leq \sqrt{\sigma_{inel}^{NN}/\pi}, \tag{6.2}$$

where σ_{inel}^{NN} is the inelastic nucleon-nucleon cross-section. This cross-section is taken to be 42 mb at 200 GeV and 40 mb at 130 GeV, and nucleons are treated as black disks.

For each binary collision, a nucleon will lose a fraction of its incident energy in the

laboratory frame. The energy loss fraction is taken as a basic model parameter. Physically, it is the mean energy lost by each nucleon in a binary collision (relative to the laboratory frame), and is assumed constant. For a binary collision of a given energy, particle production is modeled by calling a minimum bias pp collision of the same center of mass (cms) energy from the PYTHIA 6.225 event generator[28]. No additional physical processes, such as jet quenching or shadowing, are assumed. The total number of particles produced is just the sum of all particles given by all PYTHIA events.

6.2 Nucleon Kinematics

For a nucleon that has suffered n collisions, its energy in the laboratory frame is

$$E(n)_{lab} = E(0)_{lab}(1-f)^n, \tag{6.3}$$

where $E(0)_{lab}$ is the initial laboratory energy of the nucleon before undergoing any collisions, and f is the energy loss fraction. The cms energy of the collision between a nucleon that has suffered m previous collisions and another nucleon that has suffered n previous collisions is then

$$\begin{aligned} E(m,n)_{cms} &= [2(m_0^2 + E^2(0)_{lab}(1-f)^{m+n}) + 2(E^4(0)_{lab}(1-f)^{2(m+n)} \\ &- m_0^2 E^2(0)_{lab}[(1-f)^{2m} + (1-f)^{2n}] + m_0^4)^{1/2}]^{1/2}, \end{aligned} \tag{6.4}$$

where m_0 is the rest mass of a nucleon. It can be seen that there is the following condition on the number of collisions that a nucleon can suffer and still remain intact as a nucleon,

$$(1-f)^n \leq \frac{m_0}{E(0)_{lab}}. \tag{6.5}$$

The maximum number of collisions a nucleon can suffer is then

$$n_{max} = \text{floor}\left[\frac{1}{\ln(1-f)} \ln \frac{m_0}{E(0)_{lab}}\right]. \tag{6.6}$$

By requiring that the lowest amount of energy that a nucleon can have is its rest energy, the maximum number of collisions it can undergo, until its energy is no lower than its rest energy, is obtained as above. For the case of $f = 1$, a nucleon loses all of its energy in one binary collision.

In this phenomenological model, which is semi-classical, it is assumed that each nucleon can be "tracked" throughout the course of a nucleus-nucleus collision, whereas in a strictly quantum mechanical formulation, such a notion of tracking a nucleon would be ambiguous. This, however, is implicitly assumed in the Glauber model where the number of participants and binary collisions are counted. Furthermore, as a phenomenological model, it is assumed that the energy lost by a nucleon in a binary collision can be known, though in a proper quantum mechanical formulation, this is not possible. However, the choice of a constant value for the energy loss fraction yields the same results as for the case of allowing this parameter to vary randomly about a mean value. This mean value would then be equal to the constant value.

6.3 PYTHIA Settings

A discussion of the settings in PYTHIA (see Appendix B) used in the Glauber Monte Carlo program is given here[28]. The partonic transverse momentum distribution is assumed to be Gaussian, the root mean square partonic transverse momentum k_T being specified by the parameter PARP(91) in the program. To study the dependence of particle

spectra on this parameter, two scenarios are considered: one for fixed k_T and one for variable k_T. Negative hadron p_T distributions from pp collisions at $\sqrt{s} = 200$ GeV, for several values of k_T, are shown in Fig. 6.1, compared to results from the UA1 experiment[1, 29]. Although

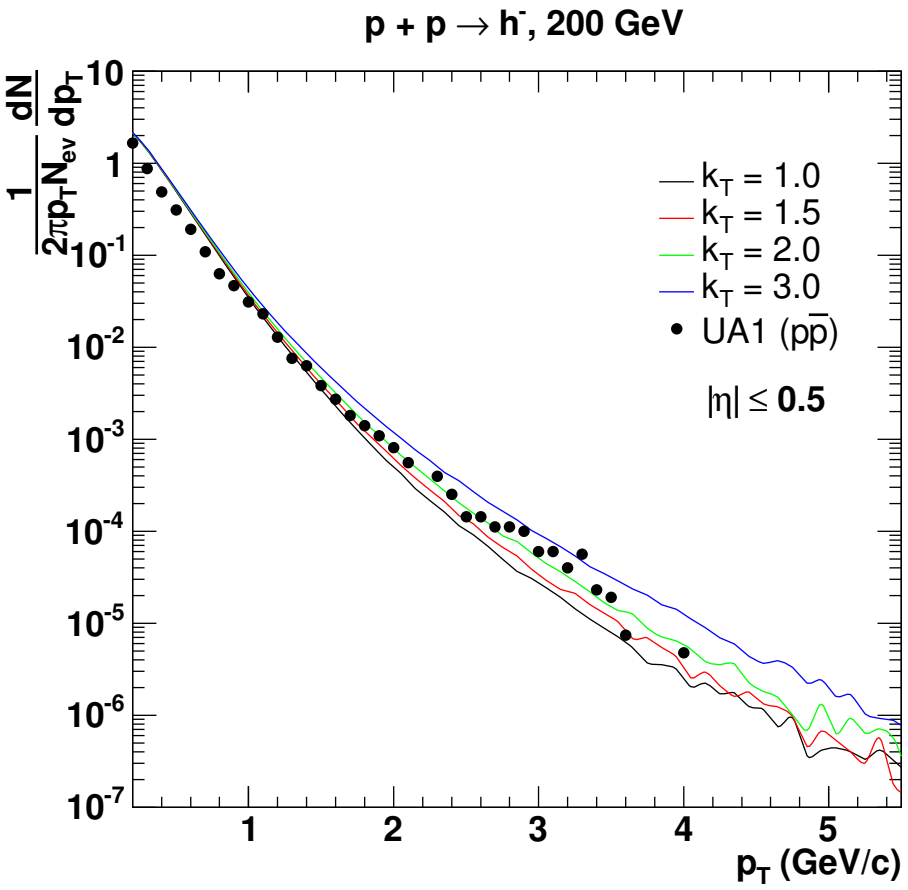

Figure 6.1: (color online) p_T distributions of negative hadrons from pp collisions at $\sqrt{s} = 200$ GeV, for several values of k_T, compared to data from the UA1 experiment[1, 29].

the distribution for $k_T = 2$ GeV/c is closest to matching the experimental data, the default value of k_T is used for the case of fixed k_T, which, in PYTHIA 6.225, is 1.0 GeV/c. This is because PYTHIA files had already been generated for $k_T = 1.0$ GeV/c, and when it was realized that the distributions with $k_T = 2.0$ GeV/c matched the experimental data

more closely, there was a time constraint on being able to generate a whole new set of files with k_T = 2.0 GeV/c. However, as shown in Fig. 6.2, the ratios of p_T distributions with k_T values differing by 1 GeV/c are seen to be approximately the same. Since ratios of p_T distributions are being calculated in this study for R_{AA} and R_{CP}, the equivalence of the ratios in Fig. 6.2 indicates that using the default value of k_T in the fixed k_T case is adequate.

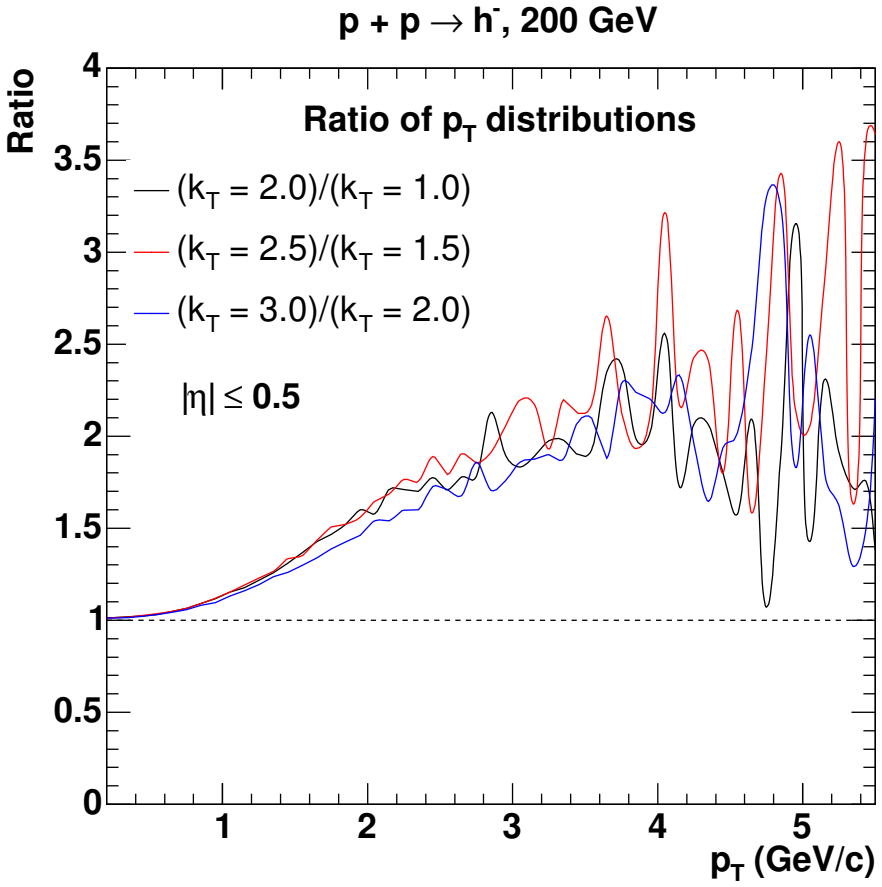

Figure 6.2: (color online) Ratios of p_T distributions with k_T values differing by 1 GeV/c, for negative hadrons from pp collisions at $\sqrt{s}$ = 200 GeV.

For variable k_T, this parameter is varied from the default value, for the highest

energy NN collisions (200 GeV), to 2.2 GeV/c for the lowest energies (10 GeV). Again, due to a time constraint, the default value of k_T was used for pp collisions at $\sqrt{s} = 200$ GeV. Lower energy pp collisions in PYTHIA correspond to multiple NN collisions in the Glauber simulation (see Section 6.5), and as such, increasing k_T at lower collision energies reproduces p_T broadening. The k_T parameter was varied so that one value of k_T corresponded to a particular range of pp collision energies, as shown in Table 6.1.

Table 6.1: Values of k_T for each energy range.

Energy Range (GeV)	k_T (GeV/c)
200	1.0
150 - 195	1.3
100 - 145	1.6
50 - 95	1.9
10 - 45	2.2

This scheme for varying k_T is not unique since the energy ranges were chosen arbitrarily. This particular scheme was divised based on its simplicity. Other schemes could also be divised that would yield similar results. The increment in the k_T parameter from one energy range to the next is taken to be 0.3 GeV/c. A higher increment would result in a greater p_T broadening and a higher peak in nuclear modification factors.

In these simulations, the target and projectile nucleon species in PYTHIA are both protons. The parton distribution function used is GRV 94L (leading order) (MSTP(51) = 4). The recommended process selection for minimum bias pp collisions in PYTHIA is used, which is MSEL = 1, for purely inelastic and non-diffractive pp collisions. Although there is no clear cut separation between soft and hard scatterings of partons, $p_T < 2$ GeV roughly corresponds to soft parton scattering and $p_T \approx 5$ GeV and higher corresponds to

hard parton scattering[30].

6.4 Implementation of PYTHIA

In this study, PYTHIA is implemented in a program called *runpythia* (see Section C.1). The inputs to this program were originally the name of the output file and the number of events to be generated. However, as this study evolved, it became necessary to also allow the center of mass energy and k_T parameter to be entered as inputs so that they could be varied automatically by a shell script. Each event is initialized by specifying the frame relative to which the collision energy is specified, the value of the collision energy, and the species of the target and projectile. The frame is chosen to be the center of mass frame, and the collision energy is then the center of mass energy. The target and projectile are both chosen to be protons.

After the initialization, the main event loop is run. For each collision, a certain process occurs, specified by a process ID number, the parameter MSTI(1). Associated with a collision are the momentum fractions, x_1 and x_2, of the two colliding partons (the variables PARI(33) and PARI(34)). These momentum fractions are used by the parton distribution function. For each collision, there are also the Mandelstam variables s, t, and u (PARI(14), PARI(15), and PARI(16), respectively), the momentum transfer scale Q^2 (PARI(22)), and the transverse momentum p_T (PARI(18)). In an event, a number of particles is produced, each specified by a particle ID number that gives the species of that particle. Each particle is also characterized by its mass and four-momentum components.

The runpythia program generates events and then writes the event information

to a *root* file which may then be processed by the ROOT software. ROOT is an object-oriented framework for data analysis [31]. The root files from runpythia are analyzed using a ROOT-based program called *readpythia* that generates histograms and profiles which, in turn, may be further manipulated in ROOT macros. When generating histograms, for example, one may limit the particles that are counted in a histogram to those particles that have transverse momentum within a certain range. The limiting of the particles that are counted in a spectrum in terms of the range of a kinematic variable is called a *kinematic cut*. A sample multiplicity distribution versus $\sqrt{s}$ is shown in Fig. 6.3 for inelastic pp collisions (MSEL = 1), for various cuts in transverse momentum.

6.5 The Glauber Monte Carlo Program

When simulating particle production in nucleus-nucleus collisions, the nuclear geometry must be specified in each collision. In the simulations carried out here, a Monte Carlo program called *glauber_mc* is used. The original program was written by Klaus Reygers of the PHENIX collaboration[32]. The program determines the numbers of binary collisions and participants in heavy ion collisions, and writes histograms for these and other geometric variables. Through a loop, the program runs a specified number of nucleus-nucleus collisions, and within this loop, a number of binary nucleon-nucleon collisions occur via a second loop. A modified form of the original program was developed to simulate particle production. A description is given here of the modified glauber_mc program. The code for this program is given in Section C.2.

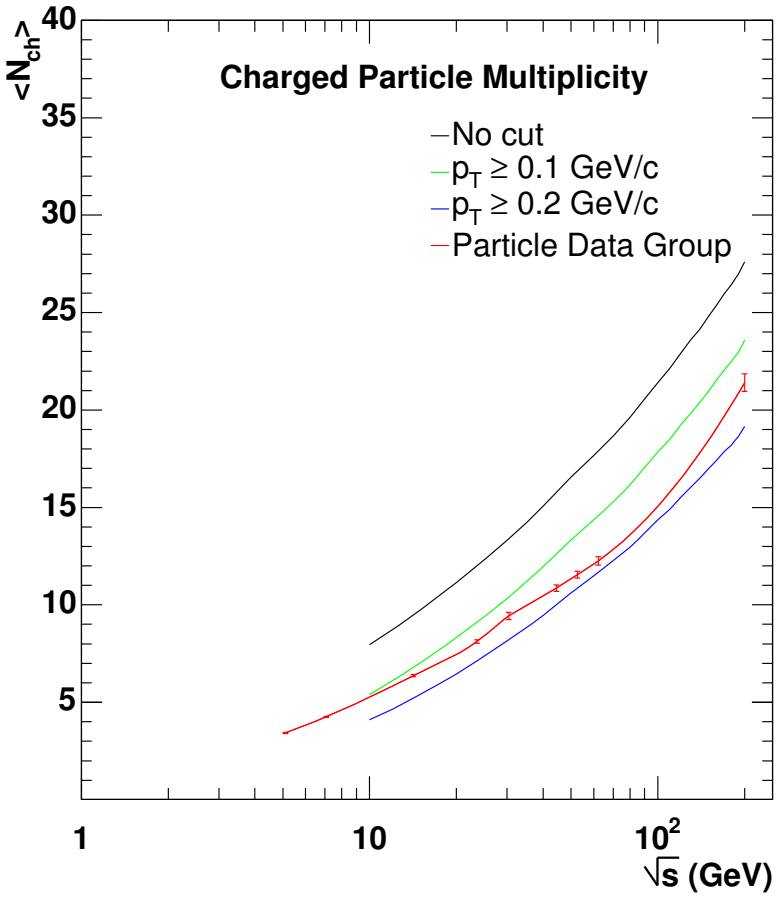

Figure 6.3: (color online) Charged particle multiplicities vs. $\sqrt{s}$ for pp collisions, for various cuts in p_T, compared to experiment[18].

6.5.1 Inputs and Outputs

The inputs to the program are read from an input file. Inputs include the number of AA collisions, the number of protons and neutrons in the target and projectile nuclei, the inelastic NN cross section in mb, the nuclear density distribution, the minimum and maximum impact parameters, the center of mass energy, and the nucleon overlap function, which determines how the nucleons overlap (for example, black disks or gray disks). Whereas the

output files of the original program only contained information on the nuclear geometry, such as distributions of the numbers of binary collisions and participants versus impact parameter, the program was modified for this study to implement a scheme by which nucleons are assumed to lose energy, with the production of particles in each binary collision. The energy loss fraction was added as an extra input, and particle production information from PYTHIA files serves as additional input. The modified program outputs information on the nuclear geometry as well as information on the produced particles, such as their ID's, energies, and momenta. Particle distributions are then calculated by a second program, *readGlauber*, that reads the output from glauber_mc.

6.5.2 The Main Algorithm

In glauber_mc, there is a loop that runs over AA collisions, and within this loop there is another one that runs over NN collisions. In the AA loop, the impact parameter for each AA collision is randomly assigned from the distribution $d\sigma/db$, as mentioned in Section 6.1. The Woods-Saxon parameters are determined by the atomic numbers of the nuclei, and from these distributions, nucleons are distributed randomly about the centers of the nuclei. One nucleus is designated as the target and the other as the projectile.

For each AA collision, a function within the program determines whether an NN collision takes place. Furthermore, the number of binary collisions that a nucleon can suffer depends on the value of the energy loss fraction. Each time an NN collision occurs, the glauber_mc program accesses particle production information from a pp collision in a file generated by PYTHIA. PYTHIA files are generated for 39 different cms energies of the pp collisions. These energies range from 10 GeV to 200 GeV in increments of 5 GeV. When an

NN collision occurs, its energy is rounded to the nearest of these 39 energies. By default, PYTHIA does not simulate pp collisions with energies lower than 10 GeV. For an NN collision with cms energy between 5 and 13 GeV, the energy is rounded to 10 GeV, and a pp collision of this energy is called from PYTHIA. For an energy between 13 and 18 GeV, the energy is rounded to 15 GeV, and so on.

The components of the four-momentum of a particle in PYTHIA are specified relative to the cms frame of the pp collision from which the particle originated. When calculating particle spectra, these components must be known relative to the laboratory frame, which is the center of mass frame of the nucleons of the *incident* nuclei, that is, of the nucleons before any collisions have occurred. A class was written for glauber_mc, called *PythiaEventAccessor*, that accesses the ID and four-momentum components of each particle from a PYTHIA pp collision, and carries out the proper Lorentz transformation from the center of mass frame of this pp collision to the laboratory frame of the AA collision. Each time a set of AA collisions is run, 39 PythiaEventAccessor objects are instantiated for each energy mentioned above. Each time an NN collision occurs, the energies and momenta of the participating nucleons, *before* the collision, are calculated so that the program will call a PYTHIA event of the appropriate collision energy.

6.6 Simulation Procedure

The procedure for running simulations of AA collisions and processing the output can be categorized into three phases:

1) The generation of PYTHIA files,

2) The generation of glauber_mc output,

3) The generation of histograms and profiles.

6.6.1 Phase 1: The Generation of PYTHIA Files

In performing simulations with glauber_mc, the PYTHIA files are generated first by running the runpythia program for each of the 39 energies. The set of pp collisions accessed by glauber_mc is divided among sets of PYTHIA files. One set of PYTHIA files consists of 39 files, one for each pp collision energy, and each file contains information from 25,000 pp collisions. Many pp collision events are needed in a file since multiple NN collisions may occur at a given energy. When simulating AA collisions, 250 AA events are simulated for each set of PYTHIA files. When simulating d + Au collisions, 2,500 d + Au events are simulated for each set of PYTHIA files. In this way, a pool of pp collisions from PYTHIA is available for access by the glauber_mc program. To reduce statistical error in calculations, a large number of AA events were simulated. For the fixed k_T case, a total of 2,000 sets of PYTHIA files were generated to allow for the simulation of 500,000 Au + Au or Cu + Cu collisions, and 5 million d + Au collisions. A second shell script was used to automate the generation of PYTHIA files for variable k_T. The total number of file sets in this case was 800, allowing 200,000 AA or 2 million d + Au collisions to be simulated. The k_T parameter was varied so that at lower energies, its value would be higher.

The PYTHIA files were generated on a cluster of computers using the *condor* management system. Condor distributed simulation jobs to nodes in the computer cluster. Using a shell script, 39 PYTHIA jobs were submitted to the condor system at a time for each energy in a set of PYTHIA files. The time needed for one set of PYTHIA files to

be generated was approximately 15 minutes. The shell script was written to allow for the automated submission of these jobs so that the files could be generated 24 hours a day for several days at a time. If the PYTHIA files were generated non-stop, 2,000 sets could be generated within 21 days, but in actuality, these sets were generated over the course of about four months, since time was taken to also run the glauber_mc program intermittently for a given number of PYTHIA sets. Also, disk drives were not always available to immediately continue the generation of these files.

The storage of the PYTHIA files required the use of several disk drives. When a disk drive ran out of space for PYTHIA files, another disk drive was added. Each set of PYTHIA files required approximately 2.1 gigabytes of memory, totaling roughly 4.2 terabytes of memory required to store the PYTHIA files for fixed k_T, and roughly 1.7 terabytes for the variable k_T files. This phase of the simulation process required the most disk space and time.

6.6.2 Phase 2: The Generation of glauber_mc Output

With sets of PYTHIA files available, the glauber_mc program could be run. As mentioned earlier, 500,000 AA events were simulated, and 5 million d + Au events were simulated for fixed k_T. All simulations were of minimum bias events. For Au + Au, the impact parameter ranged from 0 to 18 fm, for Cu + Cu the impact parameter ranged from 0 to 14 fm, and for d + Au the range was from 0 to 13 fm. For variable k_T, 2 million d + Au events were simulated.

The amount of time required to run a set of AA collisions depends on the value of the energy loss fraction and the size of the collision system. Lower values of the energy

loss fraction result in a longer run time for a given system, and heavier systems have longer run times. The run times are longer in these cases since more binary collisions occur and more particles are produced. The approximate run times in minutes, for many values of the energy loss fraction, are shown in Table 6.2 for 250 Au + Au and Cu + Cu collisions at $\sqrt{s_{NN}}$ = 200 GeV. Approximate run times for 2,500 d + Au collisions at $\sqrt{s_{NN}}$ = 200 GeV are shown in Table 6.3. The run time did not differ significantly between Au + Au at $\sqrt{s_{NN}}$ = 130 GeV and $\sqrt{s_{NN}}$ = 200 GeV. From the tables, the amount of time needed to run 2,000 sets of AA collisions on one machine ranged between roughly 4 and 6 days for Au + Au, just over a day for Cu + Cu, and about 3 days for d + Au. The full number of sets was obtained over the course of a few months as more disk drives were added to store more PYTHIA files.

Table 6.2: Run times for AA collisions.

f	Au+Au Run time (min)	Cu+Cu Run time (min)
.50	5	2
.65	4	1
.68	3	1
.70	3	1
.75	3	1
.78	2	1
1.0	below 1	below 1

Table 6.3: Run times for d + Au collisions.

f	Run time (min)
.50	3
.65	2
.68	2
.70	2
.75	2
.78	2
1.0	below 1

Approximate sizes of glauber_mc output files in megabytes, for 250 AA events, are shown in Table 6.4 for Au + Au and Cu + Cu collisions, for several values of the energy loss fraction. The sizes of glauber_mc output files for 2,500 d + Au collisions are shown in Table 6.5.

Table 6.4: File sizes for AA collisions.

f	Au+Au 130 GeV (MB)	Au+Au 200 GeV (MB)	Cu+Cu 200 GeV (MB)
.50	20	26	11
.65	20	24	9
.68	18	23	8
.70	17	19	8
.75	12	12	7
.78	10	11	7
1.0	4	6	3

Table 6.5: File sizes for d + Au collisions.

f	Filesize (MB)
.50	16
.65	13
.68	13
.70	12
.75	11
.78	11
1.0	5

6.6.3 Phase 3: The Generation of Histograms and Profiles

Histograms and profiles are generated using the readGlauber program. A shell script is employed to automate the generation of ROOT files containing histograms and profiles for each file outputted from glauber_mc. One file of histograms and profiles is typically generated within roughly 10 - 15 seconds. Then, 2,000 such files may be generated in approximately 5 - 8 hours if the program is run on one machine. Each of these files is

roughly 600 - 630 kilobytes in size for Au + Au and Cu + Cu collisions, and about 260 kilobytes for d + Au collisions.

Pseudorapidity and transverse momentum histograms were written for produced particles. Histograms and profiles were also written for geometric quantities, such as the numbers of binary collisions, participants, and the impact parameter. These histograms and profiles were then read in ROOT macros to generate plots. When writing histograms, various kinematic cuts were applied. For example, when plotting transverse momentum distributions for comparison to PHENIX data, only particles with $|\eta| < 0.18$ or $|\eta| < 0.35$ were counted for the mid-rapidity region. When plotting pseudorapidity distributions for comparison to PHOBOS data, a small fraction of particles with very low transverse momentum are not counted experimentally. However this has a negligible effect on the pseudorapidity distribution, and particles in the simulation for all transverse momenta were included.

CHAPTER 7

RESULTS

Results from the simulations are presented and compared to experiment. Distributions for the mean numbers of binary collisions and participants are presented first. Then, the obtained charged particle pseudorapidity density distributions are presented. The calculated nuclear modification factors, which are the main result of this study, are then presented and discussed.

7.1 Nuclear Geometry

The mean numbers of binary collisions and participants, $\langle N_{coll} \rangle$ and $\langle N_{part} \rangle$ respectively, for a given centrality, are dependent on the energy loss fraction. These quantities, as functions of the impact parameter b, are shown in Fig. 7.1 for Au + Au and Cu + Cu collisions at $\sqrt{s_{NN}} = 200$ GeV, for $f = .65$ and .70. This choice of values for the energy loss fraction is explained in Section 7.2. The mean number of binary collisions is determined by the maximum number of binary collisions n_{max} a nucleon can suffer, as given by Eq. 6.6

in Section 6.2. For $f = .65$, $n_{max} = 4$, and for $f = .70$, $n_{max} = 3$. Then, the distribution of the number of binary collisions will be greater in height for a lower energy loss fraction value, as more binary collisions are allowed.

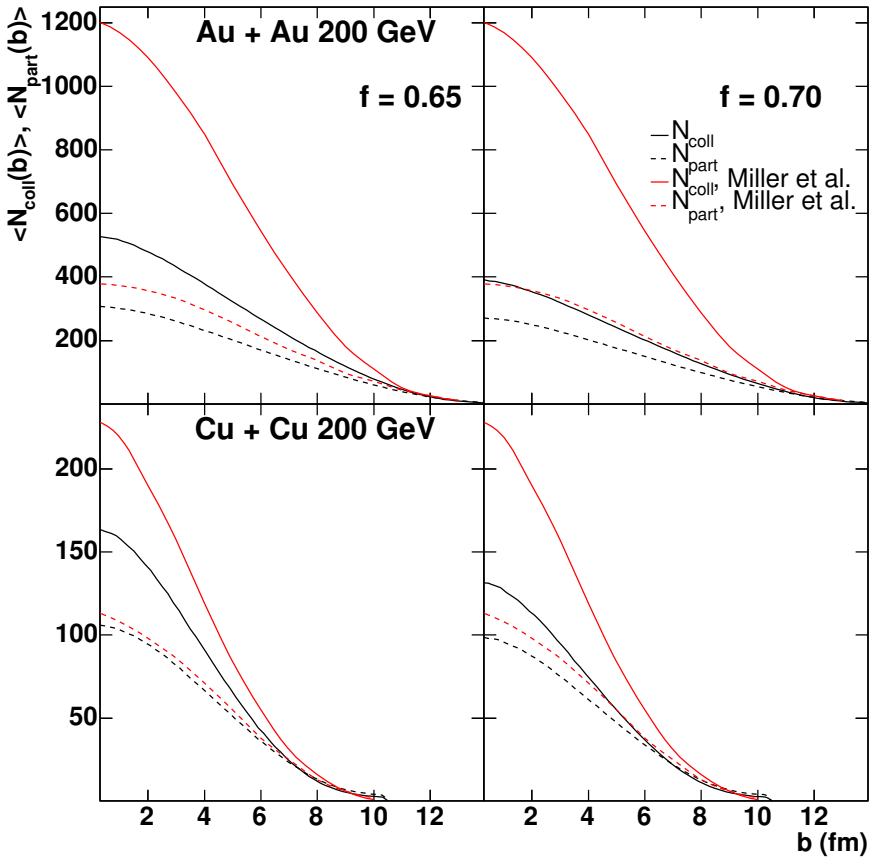

Figure 7.1: (color online) $\langle N_{coll} \rangle$ and $\langle N_{part} \rangle$ as functions of b, for Au + Au and Cu + Cu collisions at $\sqrt{s_{NN}} = 200$ GeV. The simulation results are compared to those of Miller *et al.*[20].

The mean number of participants is seen to be more weakly dependent on the energy loss fraction than is the mean number of binary collisions, which can be understood as follows. For simplicity, consider an AA collision in which the longest nucleon rows of both nuclei collide, as in Fig. 7.2. Then, the nucleons at the end of each row, as indicated

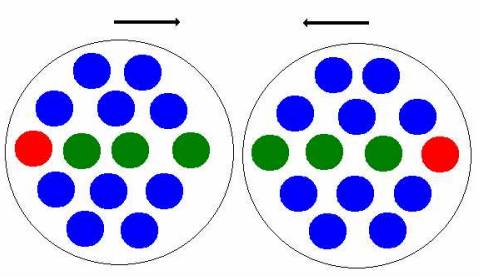

Figure 7.2: (color online) The red nucleons become participants if the green nucleons have not undergone the maximum number of binary collisions allowed for a nucleon.

by the red nucleons, will not have undergone any collisions until they meet the oncoming nucleons, as indicated by the green nucleons, which will have already undergone multiple collisions. If the green nucleons have already undergone the maximum number of binary collsions allowed for a nucleon, given by Eq. 6.6, then by the time they each meet the red nucleons (the green nucleons will all have suffered an equal number of binary collisions), the binary collisions between the red nucleons and the green nucleons will not be counted in the model simulation, and neither the number of participants nor the number of binary collisions will be incremented. However, if the green nucleons have not undergone the allowed maximum number of binary collisions, then the numbers of binary collisions and participants will both be incremented by two, since now the red nucleons will become participants. Furthermore, since the allowed maximum number of binary collisions is dependent on the energy loss fraction, the outcome of whether the red nucleons become participants, and therefore the number of participants, will also be dependent on the energy loss fraction. Row-row collisions for which some of the nucleons may or may not become participants will occur more often in more central collisions. The mean number of binary collisions is more strongly dependent on the energy loss fraction than the number of participants, since

there are more row-row collisions for which all nucleons are participants with some nucleons having reached the allowed maximum number of binary collisions, then there are row-row collisions for which not all nucleons necessarily become participants.

The mean numbers of binary collisions and participants are shown in Tables 7.1 and 7.2, respectively, for Au + Au collisions at $\sqrt{s_{NN}}$ = 200 GeV, for f = .65 and .70, compared to PHOBOS[21]. The mean numbers of binary collisions and participants are shown in Tables 7.3 and 7.4, respectively, for d + Au collisions at $\sqrt{s_{NN}}$ = 200 GeV, for the same values of f, compared to PHENIX[22].

Table 7.1: $\langle N_{coll} \rangle$ for Au + Au collisions at $\sqrt{s_{NN}}$ = 200 GeV, compared to PHOBOS calculations[21].

Centrality	f = .65	f = .70	PHOBOS
0% - 6%	454	335	1040
6% - 15%	330	246	762
15% - 25%	227	172	483
25% - 35%	151	117	286
35% - 45%	95	76	164
45% - 50%	64	53	99

Table 7.2: $\langle N_{part} \rangle$ for Au + Au collisions at $\sqrt{s_{NN}}$ = 200 GeV, compared to PHOBOS calculations[21].

Centrality	f = .65	f = .70	PHOBOS
0% - 6%	271	238	344
6% - 15%	205	179	276
15% - 25%	148	130	200
25% - 35%	104	92	138
35% - 45%	70	63	93
45% - 50%	51	46	65

For a given AA collision energy, lower values of the energy loss fraction yield higher numbers of binary collisions and participants. Moreover, the number of binary collisions and participants will also have a dependence on the collision energy of the incoming nuclei, as

Table 7.3: $\langle N_{coll} \rangle$ for d + Au collisions at $\sqrt{s_{NN}}$ = 200 GeV, compared to PHENIX calculations[22].

Centrality	f = .65	f = .70	PHENIX
0% - 20%	9.4	7.7	15.4
20% - 40%	7.8	6.6	10.6
40% - 60%	3.5	3.2	7.0
60% - 88%	1.1	1.1	3.1

Table 7.4: $\langle N_{part} \rangle$ for d + Au collisions at $\sqrt{s_{NN}}$ = 200 GeV, compared to PHENIX calculations[22].

Centrality	f = .65	f = .70	PHENIX
0% - 20%	10.5	9.0	15.0
20% - 40%	8.8	7.9	10.4
40% - 60%	4.5	4.2	7.0
60% - 88%	1.7	1.6	3.2

can be seen from Eq. 6.6. Although there would appear to be a major discrepancy between these results and those calculated by PHOBOS and PHENIX, it must be kept in mind that the numbers of binary collisions and participants are model dependent quantities and are not measured experimentally. Glauber Monte Carlo simulations are also used by PHOBOS and PHENIX to calculate these quantities, but without the assumption of nucleon energy loss.

7.2 Pseudorapidity Density Distributions

In the charged particle spectra, the particles included are pions, kaons, protons, and antiprotons. The pseudorapidity density distributions for charged particles for Au + Au collisions at $\sqrt{s_{NN}}$ = 130 and 200 GeV, and Cu + Cu collisions at $\sqrt{s_{NN}}$ = 200 GeV, are shown in Fig. 7.3. The pseudorapidity distribution is used to determine values of the

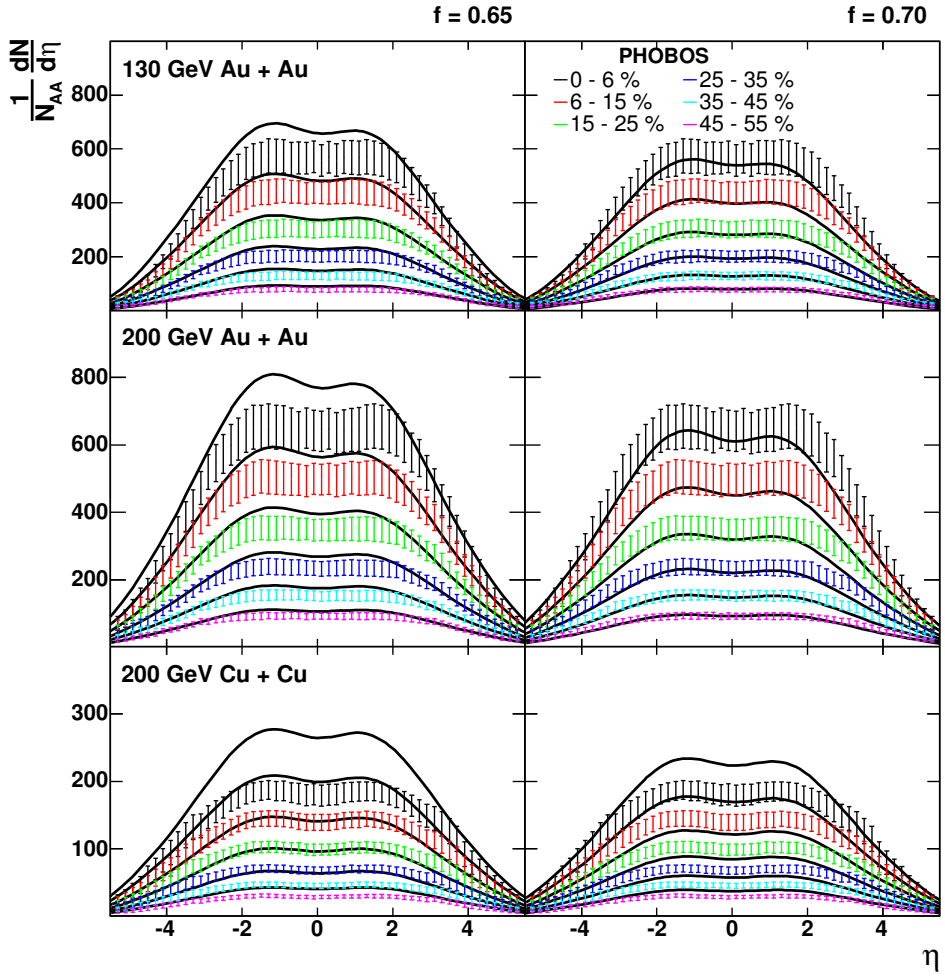

Figure 7.3: (color online) Charged particle pseudorapidity density distributions for Au + Au collisions at $\sqrt{s_{NN}}$ = 130 and 200 GeV, and Cu + Cu collisions at $\sqrt{s_{NN}}$ = 200 GeV, compared to PHOBOS data[21, 33]. The spectra from the simulation are indicated by the solid black curves for each centrality.

energy loss fraction. In fitting these distributions to experimental data, it is found that values of the energy loss fraction of f = .65 - .70 reproduce the distributions quite well for Au + Au. Lower values of the energy loss fraction result in higher yields. This is due to the larger number of binary collisions that occur for slower depletion of the nucleon energies at lower values of the energy loss fraction. The distributions with the higher value of the energy loss fraction match more closely with experiment for mid-rapidity, and the distributions with the lower value match more closely at larger rapidity. However, for these values of the energy loss fraction, the corresponding Cu + Cu distributions overshoot the experimental data at mid-rapidity. Higher energy loss fractions would better fit the Cu + Cu data. The energy loss fraction is dependent on nuclear size.

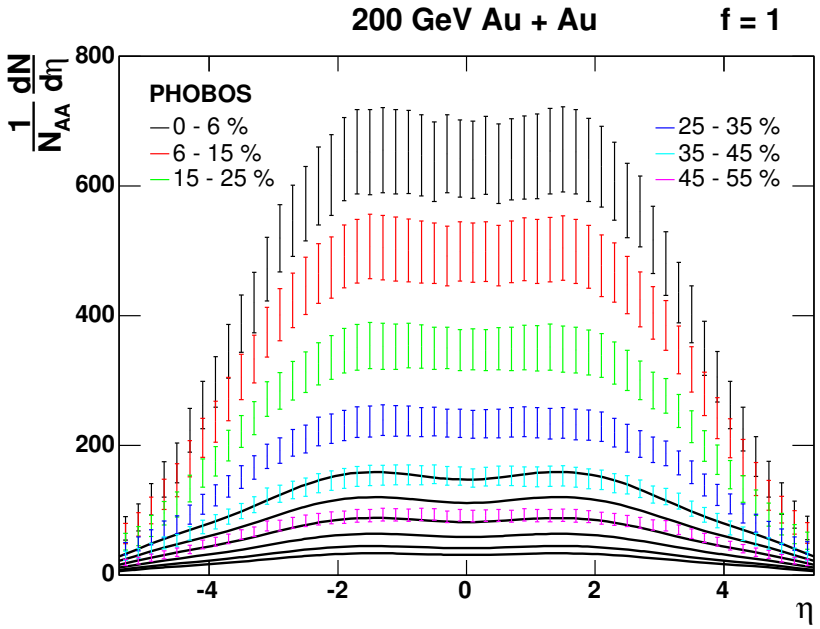

Figure 7.4: (color online) Charged particle pseudorapidity distributions for f = 1, for Au + Au collisions at $\sqrt{s_{NN}}$ = 200 GeV, compared to PHOBOS data[21]. The spectra from the simulation are indicated by the solid black curves for each centrality.

The charged particle pseudorapidity distributions for f = 1, for Au + Au collisions

at $\sqrt{s_{NN}}$ = 200 GeV, are shown in Fig. 7.4. Particle production is minimal in this case since each nucleon can only undergo one collision. The charged particle pseudorapidity distributions for d + Au collisions at $\sqrt{s_{NN}}$ = 200 GeV are shown in Fig. 7.5, and are seen to be comparable to the PHOBOS data for the same range of values of the energy loss fraction as those of Au + Au. Furthermore, the pseudorapidity density distributions for variable k_T are found to differ very little from the distributions for fixed k_T.

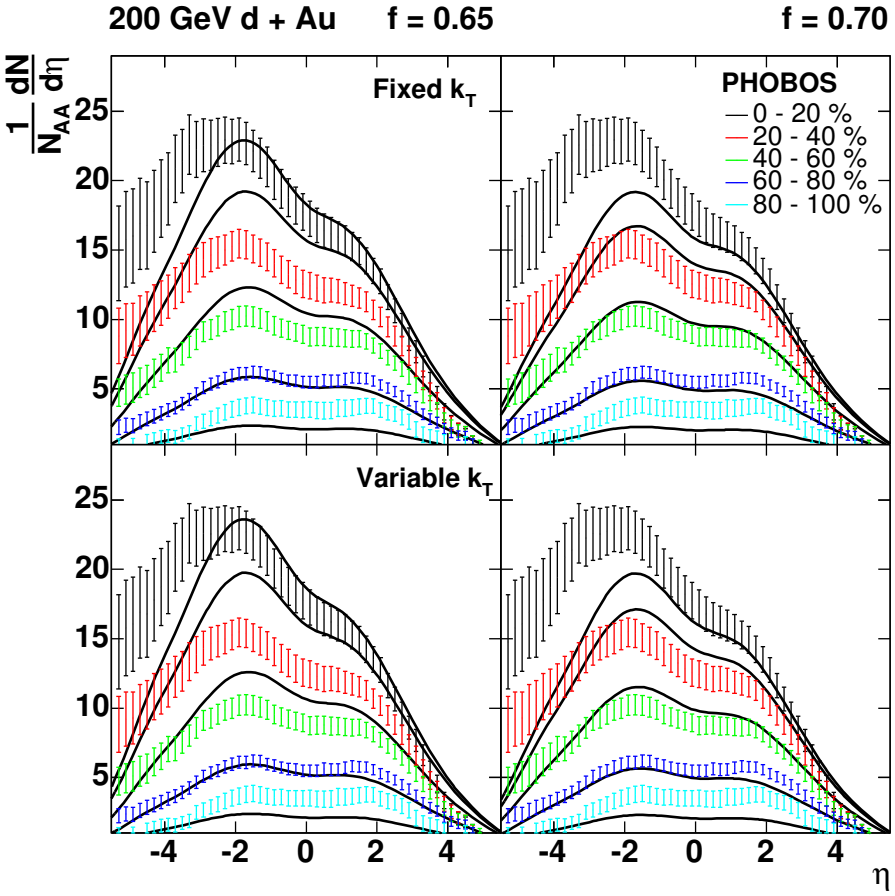

Figure 7.5: (color online) Comparison of charged particle pseudorapidity density distributions for d + Au collisions at $\sqrt{s_{NN}}$ = 200 GeV, for fixed and variable k_T. Results, indicated by the solid black curves, are compared to PHOBOS data[23].

7.3 Nuclear Modification Factors

The nuclear modification factor R_{AA} for charged particles is shown in Fig. 7.6 for Au + Au collisions at $\sqrt{s_{NN}}$ = 200 GeV. The calculated spectra are closer to the experimental data for the most peripheral collisions, where nuclear medium effects diminish. The π^0 R_{AA}'s for Au + Au and Cu + Cu collisions at $\sqrt{s_{NN}}$ = 200 GeV are shown in Figs.

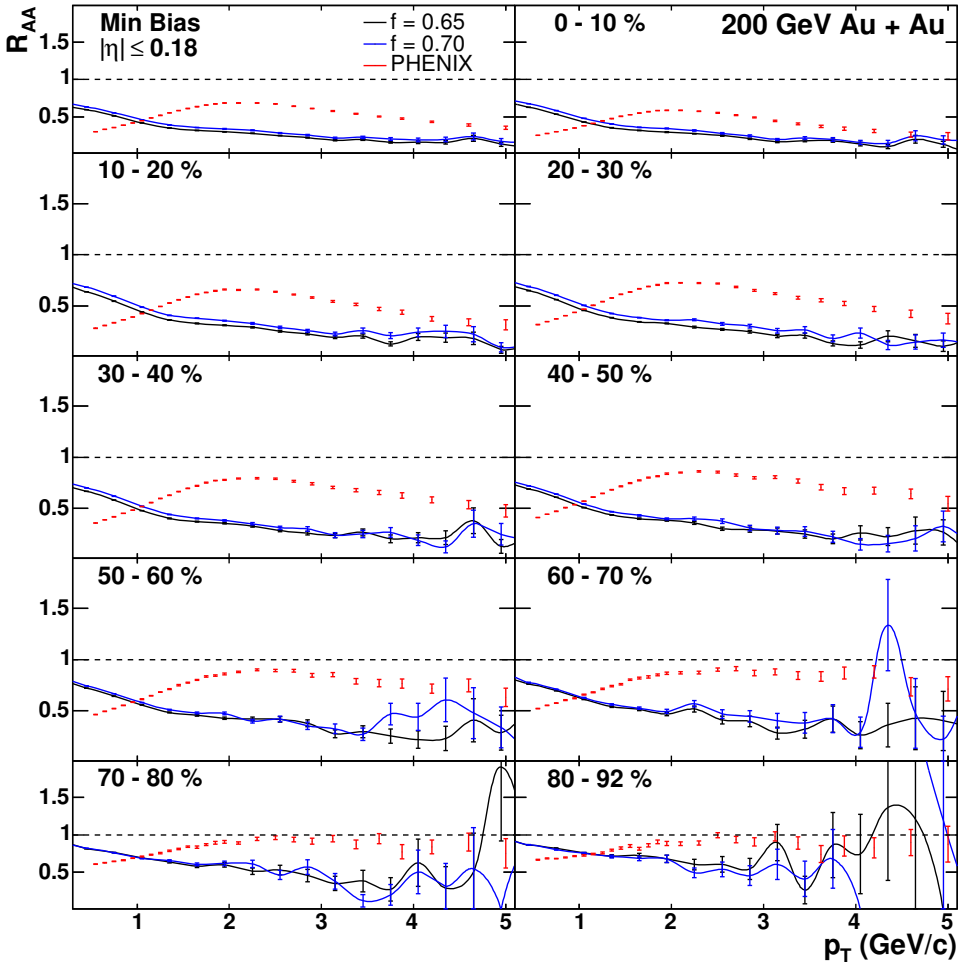

Figure 7.6: (color online) Charged particle R_{AA} for Au + Au collisions at $\sqrt{s_{NN}}$ = 200 GeV, compared to PHENIX data[24]. The calculated spectra are indicated by the solid curves.

7.7 and 7.8, respectively. They are seen to follow the same trend with centrality as the

charged particles. The charged particle R_{CP} is shown in Fig. 7.9 for Au + Au at $\sqrt{s_{NN}} =$ 130 and 200 GeV. All errors of calculated nuclear modification factors are statistical. Below

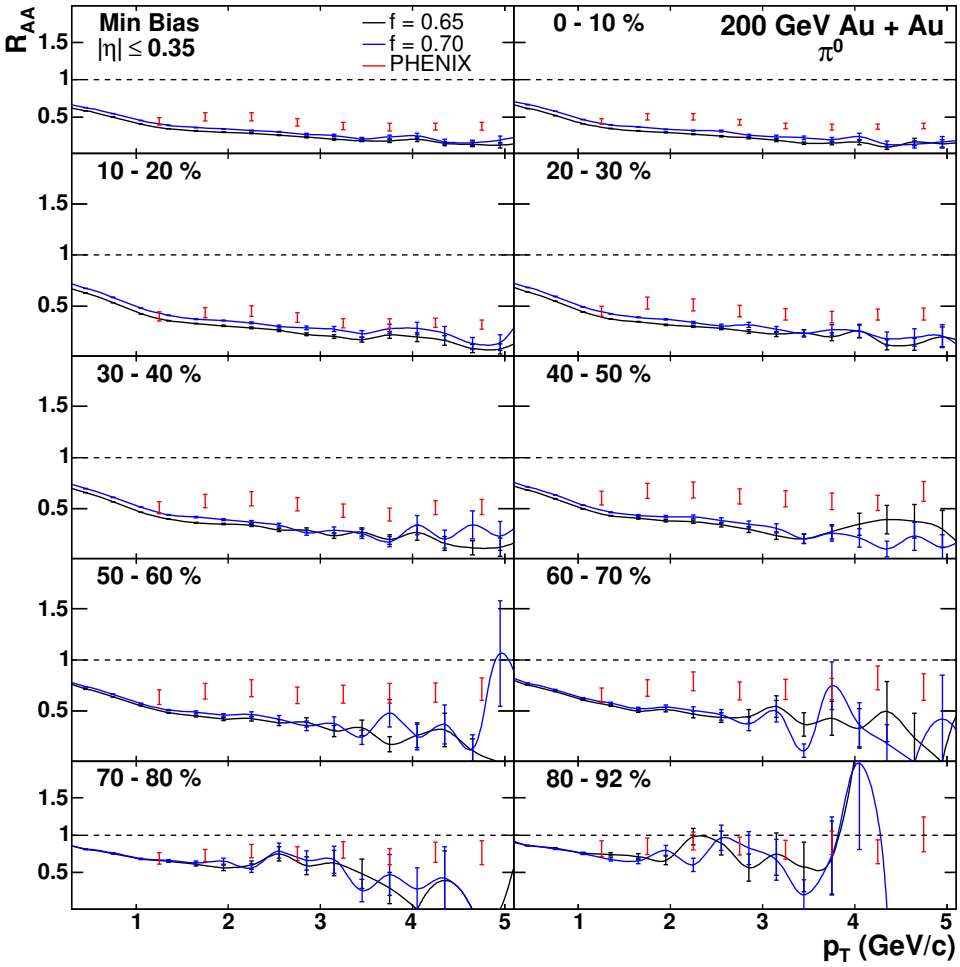

Figure 7.7: (color online) π^0 R_{AA} for Au + Au collisions at $\sqrt{s_{NN}} = 200$ GeV, compared to PHENIX data[24]. The calculated spectra are indicated by the solid curves.

$p_T \approx 1$ GeV/c, the curves for the simulated nuclear modification factors are higher than the experimental data. This is due to a limitation in the modeling of soft processes in PYTHIA. The nuclear modification factors are seen to be less than unity and to depend very weakly on the energy loss fraction.

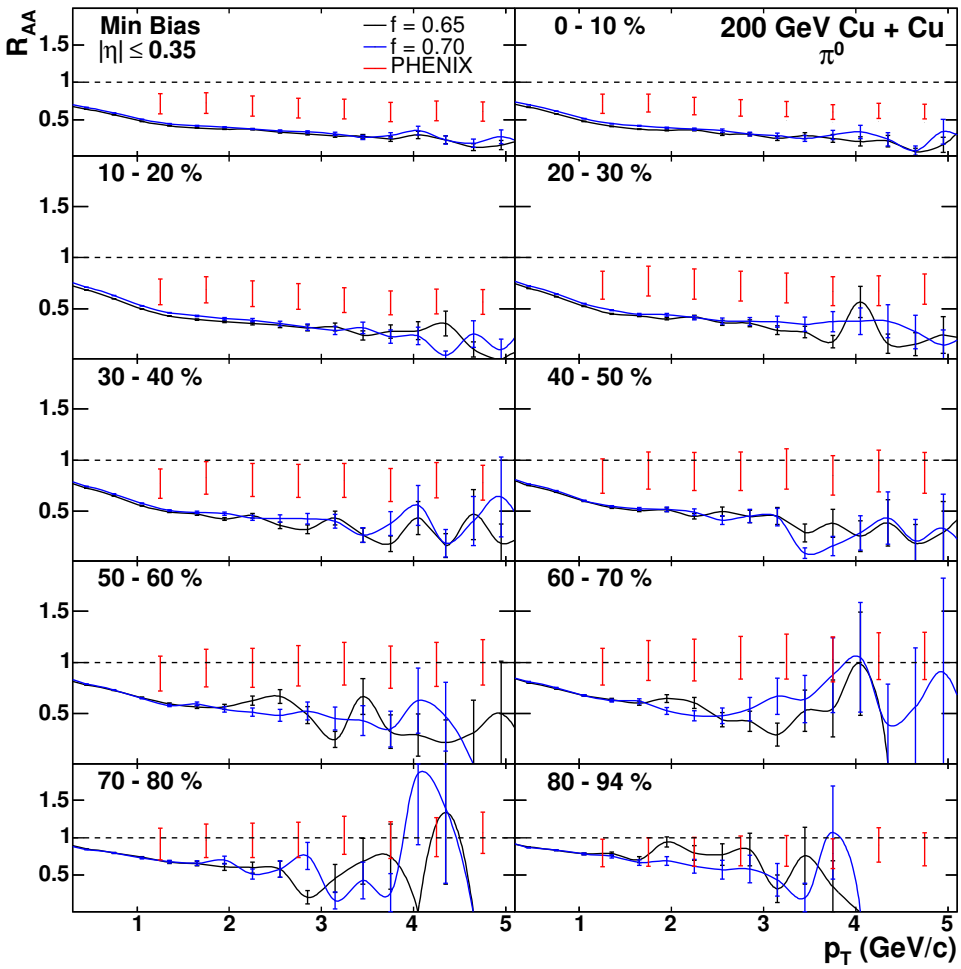

Figure 7.8: (color online) π^0 R_{AA} for Cu + Cu collisions at $\sqrt{s_{NN}}$ = 200 GeV, compared to PHENIX data[34]. The calculated spectra are indicated by the solid curves.

The charged particle R_{dAu} distributions are shown in Fig. 7.10 for fixed and variable k_T. For the fixed k_T case, as shown in the top panels of Fig. 7.10, k_T = 1.0 GeV/c, and in the variable k_T case, shown in the bottom panels of Fig. 7.10, k_T ranges from 1.0 to 2.2 GeV/c. For nucleons that have undergone multiple collisions, as their energy is lowered in each subsequent collision, the k_T parameter for each corresponding collision in PYTHIA is raised. Varying k_T in this way reproduces the p_T broadening of particle

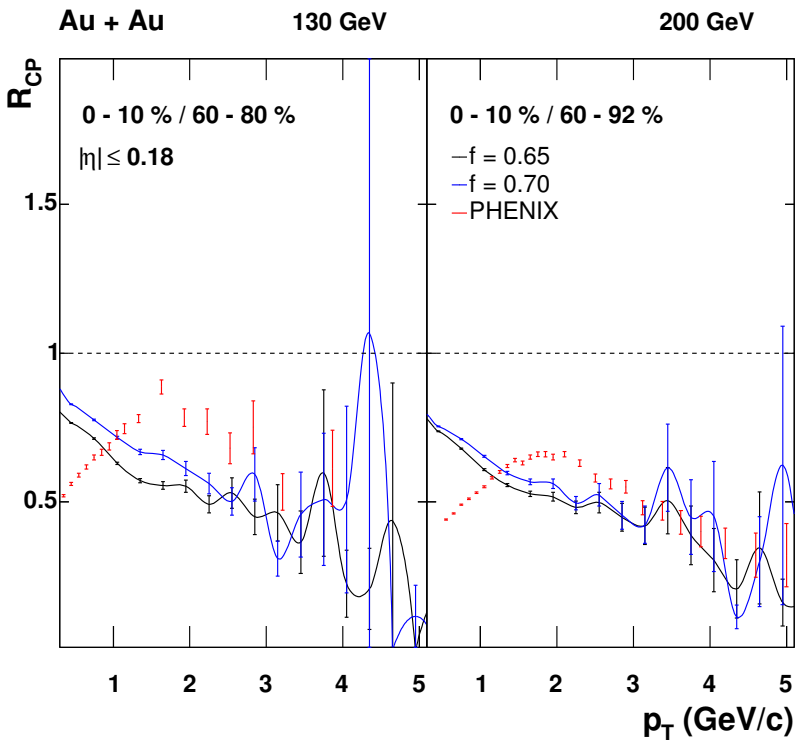

Figure 7.9: (color online) Charged particle R_{CP} for Au + Au at $\sqrt{s_{NN}} = 130$ and 200 GeV, compared to PHENIX data[9, 24]. The calculated spectra are indicated by the solid curves.

spectra from multiple nucleon collisions. The nuclear modification factors are again seen to depend very weakly on the energy loss fraction. Varying the k_T parameter results in only a minor variation of the pseudorapidity density distributions as mentioned before, but in a significant variation in R_{dAu} as a function of p_T. For fixed k_T, the calculated R_{dAu} distributions are less than unity at high p_T, and for variable k_T, they approach or surpass unity for $p_T > 2$ GeV/c, apart from statistical fluctuations. These results indicate that for incoherent binary collisions with nucleon energy loss, R_{dAu} does not equal or exceed unity unless p_T broadening is taken into account.

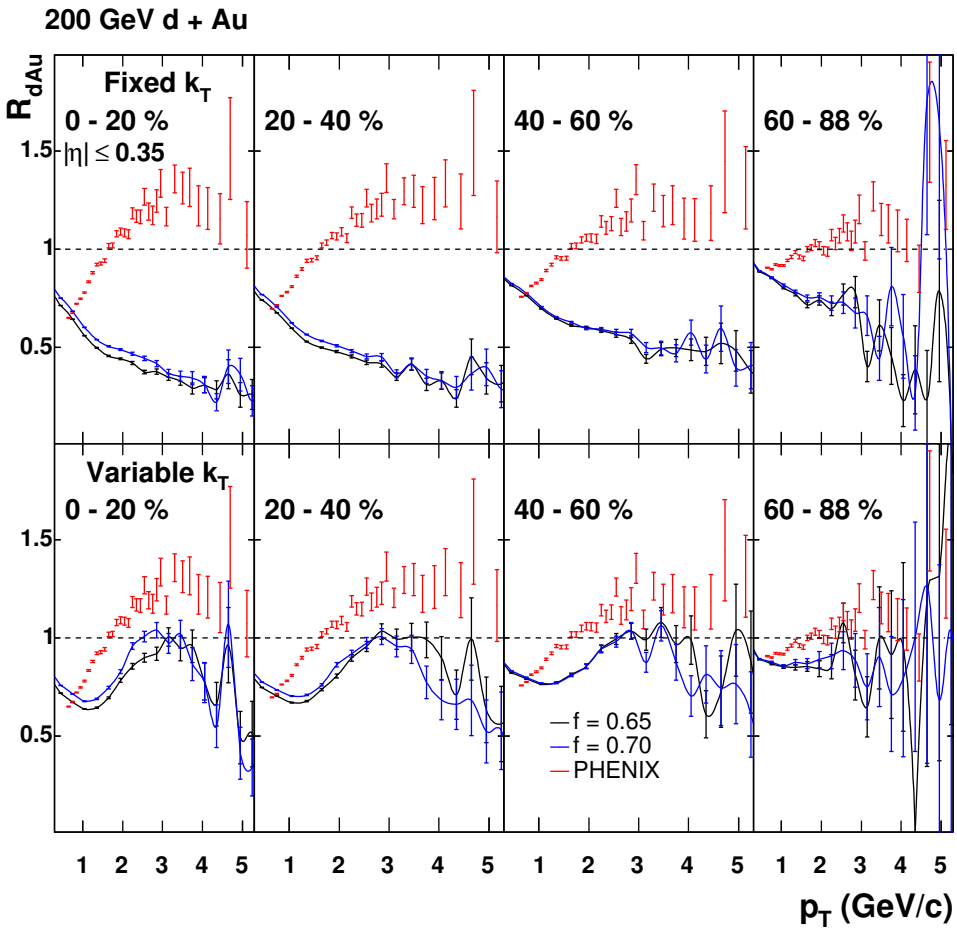

Figure 7.10: (color online) Charged particle R_{dAu} for d + Au collisions at $\sqrt{s_{NN}} = 200$ GeV. The top panels show the results for fixed k_T, and the lower panels show those for variable k_T. The calculated spectra, indicated by the solid curves, are compared to PHENIX data[22].

7.4 Discussion of the Nuclear Modification Factors

No nuclear medium is assumed to be produced in this model of incoherent binary collisions. All particle production results simply from binary collisions of nucleons. It is seen that, although there is not a close match overall, apart from statistical error, the calculated R_{AA} and R_{CP} spectra are comparable to those of experiment at higher p_T.

The R_{dAu} spectra are comparable to experiment if the k_T parameter is varied for

nucleons that have suffered multiple collisions. Moreover, it is commonly thought that in the case of no nuclear medium or other modification effects, $R_{AA} = 1$ for hard contributions to particle yields. However, in the results for fixed k_T, it is seen that R_{AA} is certainly less than unity, and this may be attributed to the energy loss in nucleon-nucleon collisions and the manner in which these nuclear modification factors are defined. The pp reference spectrum in the denominator of R_{AA} is that of pp collisions with cms energy equal to the initial (maximum) cms energy per nucleon pair of the colliding nuclei. On the other hand, the spectrum in the numerator of R_{AA} consists of contributions from a mixture of NN collisions with energies less than or equal to the cms energy of the pp reference spectrum. Then, with the pp spectrum multiplied by the mean number of binary collisions, the scaled pp spectrum in the denominator of R_{AA} will be greater in magnitude than the incoherent spectrum in the numerator, and R_{AA} will necessarily be less than unity. Furthermore, only with p_T broadening does R_{dAu} approach or surpass unity when nucleon energy loss is taken into account. The degree of p_T broadening is not quantified by how much R_{dAu} exceeds unity since this broadening effect is needed for R_{dAu} to equal unity in the first place. The nuclear modification factors R_{AA}, R_{dAu}, and R_{CP} do not quantify nuclear effects simply by differing from unity at high p_T.

CHAPTER 8

SUMMARY AND CONCLUSIONS

An incoherent binary nucleon-nucleon collision model of nuclear collisions is presented for simulating particle production in cold nuclear matter using a Glauber Monte Carlo approach. Particle production is simulated using the PYTHIA 6.2 event generator. With a simple phenomenological parameter, the mean nucleon energy loss fraction, results are obtained for the pseudorapidity density distributions and nuclear modification factors of Au + Au, Cu + Cu, and d + Au collisions. The pseudorapidity density distributions have a strong dependence on the energy loss fraction, while the nuclear modification factors depend strongly on the k_T parameter. The obtained particle distributions are comparable to experimental data. Although there is not a close match to the experimental data overall, apart from statistical error, the calculated R_{AA} and R_{CP} are shown to be less than unity due to nucleon energy loss. This is in contrast to the commonly held view that they should

equal unity at high p_T, in the absence of any initial or final state nuclear effects, when treating an AA collision as a sum of incoherent binary collisions. The nuclear modification factor R_{dAu} is also shown to be less than unity unless the k_T parameter is varied to give p_T broadening. Further simulations should be carried out to study p_T broadening in heavy systems, such as Au + Au and Cu + Cu, by varying the k_T parameter.

The usual definitions of the nuclear modification factors do not take into account the energy loss of nucleons. They do not quantify nuclear effects simply by differing from unity at high p_T. Furthermore, although only one hard parton collision is likely per binary collision, as nucleons undergo multiple collisions and lose energy, the corresponding hard collisions should also occur with lower energies. From these considerations, particle yields from hard processes do not scale with the number of binary collisions in the incoherent case. Therefore, it is concluded that the nuclear modification factors, as they are defined, are insufficient for quantifying effects on particle production resulting from a produced medium.

BIBLIOGRAPHY

[1] V. Topor Pop *et al.*, Phys. Rev. C **68**, 054902 (2003).

[2] X. N. Wang and M. Gyulassy, Phys. Rev. D **44**, 3501 (1991).

[3] V. Topor Pop *et al.*, Phys. Rev. C **70**, 064906 (2004).

[4] S. Jeon and J. Kapusta, nucl-th/9703033.

[5] C. Y. Wong and Z. D. Lu, Phys. Rev. D **39**, 2606 (1989).

[6] X. N. Wang and M. Gyulassy, Phys. Rev. Lett. **68**, 1480 (1992).

[7] B. Alver *et al.* (PHOBOS Collaboration), Phys. Rev. Lett. **96**, 212301 (2006).

[8] S. S. Adler *et al.* (PHENIX Collaboration), Phys. Rev. Lett. **96**, 202301 (2006).

[9] K. Adcox *et al.* (PHENIX Collaboration), Phys. Rev. Lett. **88**, 022301 (2002).

[10] C. Y. Wong, *Introduction to High-Energy Heavy-Ion Collisions*, World Scientific Publishing Company, Singapore, 1994.

[11] http://pdg.lbl.gov/2007/tables/contents_tables.html

[12] R. Zaballa, *Quarkonia as Probes of the Quark-Gluon Plasma*, Master's Degree Presentation, Georgia State University (2006).

[13] M. Guidry, *Gauge Field Theories: An Introduction with Applications*, John Wiley & Sons, Inc., New York, 1991.

[14] F. E. Close, *An Introduction to Quarks and Partons*, Academic Press, London, 1979.

[15] K. Yagi, T. Hatsuda, and Y. Miake, *Quark-Gluon Plasma: From Big Bang to Little Bang*, Cambridge University Press, Cambridge, 2005.

[16] http://www.ice.csic.es/es/graphics/

[17] http://ipap.yonsei.ac.kr/~npl/intro0.html

[18] http://pdg.lbl.gov/2005/reviews/contents_sports.html#kinemaetc

[19] K. Adcox *et al.* (PHENIX Collaboration), Nucl. Phys. A **757**, 184 (2005).

[20] M. L. Miller *et al.*, nucl-ex/0701025v1.

[21] B. B. Back *et al.* (PHOBOS Collaboration), Phys. Rev. Lett. **91**, 052303 (2003).

[22] S. S. Adler *et al.* (PHENIX Collaboration), Phys. Rev. C **77**, 014905 (2008).

[23] B. B. Back *et al.* (PHOBOS Collaboration), Phys. Rev. C **72**, 031901(R) (2005).

[24] S. S. Adler *et al.* (PHENIX Collaboration), Phys. Rev. C **69**, 034910 (2004).

[25] S. S. Adler *et al.* (PHENIX Collaboration), Phys. Rev. Lett. **98**, 172302 (2007).

[26] J. W. Cronin *et al.*, Phys. Rev. D **11**, 3105 (1975).

[27] X. F. Zhang and G. Fai, hep-ph/0306227.

[28] T. Sjöstrand *et al.*, hep-th/0108264.

[29] C. Albajar *et al.*, Nucl. Phys. **B335**, 261 (1990).

[30] http://www.phys.ufl.edu/~rfield/cdf/chgjet/pythia.html

[31] http://root.cern.ch/

[32] https://www.phenix.bnl.gov/phenix/WWW/publish/chang/e917/glauber_mc.cpp

[33] B. Alver *et al.* (PHOBOS Collaboration), "System Size, Energy and Centrality Dependence of Pseudorapidity Distributions of Charged Particles in Relativistic Heavy Ion Collisions", Submitted for publication in Phys. Rev. Lett.

[34] A. Adare *et al.* (PHENIX Collaboration), nucl-ex/0801.4555v1.

[35] J. D. Jackson, *Classical Electrodynamics*, 3rd edn., John Wiley & Sons, Inc., 1999.

APPENDIX A: RELATIVISTIC KINEMATICS

The kinematics of special relativity, necessary for understanding heavy-ion collisions and other processes, is introduced here. The notations of Wong and Jackson[10, 35] will be used, and all work will be presented in *natural units*, $c = \hbar = 1$. Four-vectors and Lorentz transformations are first introduced, followed by kinematic variables and Lorentz-invariant cross sections.

A.1 Four-vectors and Lorentz Transformations

Space and time together form a geometric continuum consisting of four dimensions, and vectors in space-time are referred to as *four-vectors*. Four-vectors will be denoted simply by letters, such as a and p, and vectors in three-dimensional space will be denoted by letters with arrows, such as $\vec{a}$ and $\vec{p}$. The coordinates of a point x in space-time form a four-vector relative to the origin with components x^μ, where the index μ takes on the values $\mu = 0, 1, 2, 3$

for each component,

$$x^\mu = \left(x^0, x^1, x^2, x^3\right) = (t, \vec{x}) = (t, x, y, z)\,, \tag{A.1.1}$$

and similarly, a particle's momentum four-vector may be expressed in terms of its components p^μ,

$$p^\mu = \left(p^0, p^1, p^2, p^3\right) = (E, \vec{p}) = (E, p_x, p_y, p_z)\,. \tag{A.1.2}$$

Consider two inertial reference frames, S and S', in relative motion such that at $x^0 = x'^0 = 0$, their origins coincide, and frame S' moves along the positive z-axis, with velocity β, relative to frame S, as shown in Fig. A.1.

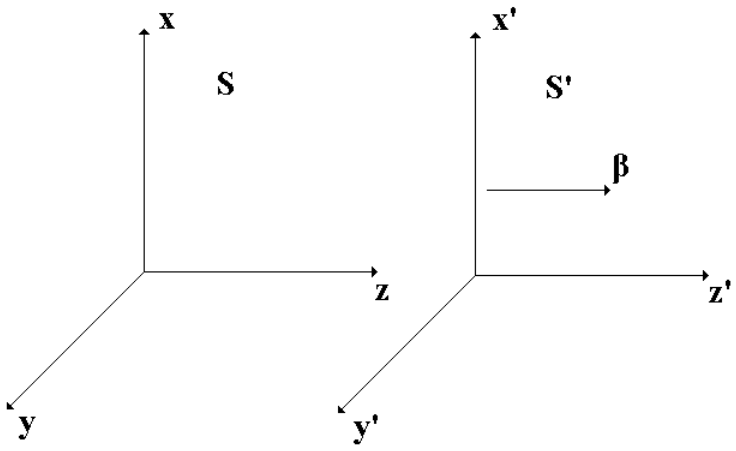

Figure A.1: Inertial frames in relative motion.

The coordinates defined relative to frame S' will be given in terms of those defined

relative to frame S by the Lorentz transformation,

$$\begin{aligned} x'^0 &= t' = \gamma\left(x^0 - \beta x^3\right) = \gamma\left(t - \beta z\right) \\ x'^1 &= x' = x^1 = x \\ x'^2 &= y' = x^2 = y \\ x'^3 &= z' = \gamma\left(x^3 - \beta x^0\right) = \gamma\left(z - \beta t\right), \end{aligned} \tag{A.1.3}$$

where $\gamma = 1/\sqrt{1-\beta^2}$, and the inverse of this transformation is

$$\begin{aligned} x^0 &= t = \gamma\left(x'^0 + \beta x'^3\right) = \gamma\left(t' + \beta z'\right) \\ x^1 &= x = x'^1 = x' \\ x^2 &= y = x'^2 = y' \\ x^3 &= z = \gamma\left(x'^3 + \beta x'^0\right) = \gamma\left(z' + \beta t'\right). \end{aligned} \tag{A.1.4}$$

Other four-vectors similarly transform so that, for example, the momentum four-vector p transforms as

$$\begin{aligned} p'^0 &= E' = \gamma\left(p^0 - \beta p^3\right) = \gamma\left(E - \beta p_z\right) \\ p'^1 &= p'_x = p^1 = p_x \\ p'^2 &= p'_y = p^2 = p_y \\ p'^3 &= p'_z = \gamma\left(p^3 - \beta p^0\right) = \gamma\left(p_z - \beta E\right), \end{aligned} \tag{A.1.5}$$

with inverse

$$\begin{aligned} p^0 &= E = \gamma\left(p'^0 + \beta p'^3\right) = \gamma\left(E' + \beta p'_z\right) \\ p^1 &= p_x = p'^1 = p'_x \\ p^2 &= p_y = p'^2 = p'_y \\ p^3 &= p_z = \gamma\left(p'^3 + \beta p'^0\right) = \gamma\left(p'_z + \beta E'\right). \end{aligned} \tag{A.1.6}$$

A consequence of the Lorentz transformation is the *Lorentz contraction* of lengths parallel to an object's direction of motion. Consider a rod of length ℓ' at rest with respect to reference frame S' and parallel to the z'-axis. Spatial length is the separation between two space-time points that have differing spatial coordinates but have the same time coordinate relative to a particular reference frame. Therefore, since the length of the rod in reference frame S' is $\ell' = \Delta z'$, then according to the Lorentz transformation equations, the length of the rod, ℓ, relative to S is related to the rod's length relative to S' by

$$\ell' = \Delta z' = \gamma\left(\Delta z - \beta \Delta t\right) = \gamma \Delta z = \gamma \ell, \tag{A.1.7}$$

or

$$\ell = \ell'/\gamma = \ell'\sqrt{1-\beta^2}, \tag{A.1.8}$$

so that the length of the rod relative to S is shorter in comparison to the rod's length relative to S'. Conversely an identical rod at rest relative to S will have a shortened length relative to S'.

Another consequence of the Lorentz transformation is *time dilation*. Consider a clock at rest relative to S'. The ticks of this clock correspond to space-time points with differing time coordinates, but having the same spatial coordinates relative to S'. Relative

to S these space-time points will have differing time and spatial coordinates. If the amount of time between two successive ticks of this clock is $\Delta t'$ relative to S', then relative to S the amount of time that passes between these same two ticks is given by

$$\Delta t = \gamma \left(\Delta t' + \beta \Delta z' \right) = \gamma \Delta t', \tag{A.1.9}$$

and relative to frame S this clock runs slower than a clock at rest relative to S. Conversely an observer in S' will observe a clock, at rest relative to S, to run slower than clocks at rest relative to frame S'.

A third consequence of the Lorentz transformation is that velocities do not add as they do in Newtonian mechanics. Consider a particle moving with velocity $v' = dx'^3/dx'^0$ along the z'-axis relative to S'. The velocity of this particle relative to S is then $v = dx^3/dx^0$ and from the Lorentz transformation, the velocities of the particle relative to each of these frames are related by

$$v = \frac{dx^3}{dx^0} = \frac{\gamma \left(dx'^3 + \beta dx'^0 \right)}{\gamma \left(dx'^0 + \beta dx'^3 \right)} = \frac{dx'^0}{dx'^0} \frac{dx'^3/dx'^0 + \beta}{1 + \beta dx'^3/dx'^0} = \frac{v' + \beta}{1 + \beta v'}. \tag{A.1.10}$$

For any velocity v, $0 \leq v < 1$, and from this relation, which gives the addition of the velocities β and v', it is seen that their sum can never exceed unity (the speed of light).

The scalar product of two four-vectors, a and b is given by

$$a \cdot b = a^0 b^0 - \vec{a} \cdot \vec{b}, \tag{A.1.11}$$

and the magnitude of a four-vector by

$$|a| = \sqrt{(a^0)^2 - |\vec{a}|^2}. \tag{A.1.12}$$

Scalar products and magnitudes of four-vectors are invariant under Lorentz transformations since they are scalar quantities.

The momentum of a particle is given in terms of its velocity v and mass m by

$$\vec{p} = \gamma m \vec{v}, \tag{A.1.13}$$

where $\gamma = 1/\sqrt{1-v^2}$, and its total energy by

$$E = \gamma m. \tag{A.1.14}$$

From this equation for a particle's total energy, it is seen that when a particle is at rest ($\vec{v} = \vec{0}$), it will have a rest energy equal to its mass. The kinetic energy of a particle is given by

$$E = (\gamma - 1)\, m, \tag{A.1.15}$$

and the magnitude of the four-momentum is

$$|p| = \sqrt{E^2 - \vec{p} \cdot \vec{p}} = m. \tag{A.1.16}$$

A.2 Kinematic Variables

In the description of relativistic processes, several variables may be defined. Consider an interaction between two particles a and b, where particle a is referred to as the *projectile* and particle b as the *target* so that particle a is incident on particle b. The reference frame in which particle b is at rest, with particle a incident upon it, will be referred to as the *laboratory frame*. In this frame then, $\vec{p}_{b,lab} = 0$ and $E_{b,lab} = m_b$, where m_b is the target particle's rest mass. The two particles collide, represented by the reaction

$$a + b \rightarrow c + d + e + \cdots, \tag{A.2.17}$$

where the final product on the right-hand side consists of many particles. The reaction is said to have a two-body *initial state* and a many-body *final state*[15]. Two particles collide and after the collision, many particles come out of the collision. In the *center of mass frame*, the two incoming particles have equal but opposite momenta so that their total momentum is zero, and the total momentum of all particles coming out of the collision is also zero,

$$\vec{p}_{a,cm} + \vec{p}_{b,cm} = \vec{p}_{c,cm} + \vec{p}_{d,cm} + \vec{p}_{e,cm} + \cdots = 0 \tag{A.2.18}$$

For a reaction with two-body initial and final states, $a+b \to c+d$, the *Mandelstam variables*, which are Lorentz invariant quantities, are given by[13]

$$s \equiv |p_a + p_b|^2, \tag{A.2.19}$$

$$t \equiv |p_a - p_c|^2, \tag{A.2.20}$$

and

$$u \equiv |p_a - p_d|^2. \tag{A.2.21}$$

The variable s is convenient for characterizing the total energy of a collision that may result in a many-body final state. In the laboratory frame, $p_{a,lab} = (E_{lab}, \vec{p}_{lab})$, $p_{b,lab} = \left(m, \vec{0}\right)$, and

$$\begin{aligned} s &= |p_a + p_b|^2 = |p_{a,lab} + p_{b,lab}|^2 \\ &= (E_{lab} + m)^2 - (\vec{p}_{lab})^2 = (E_{lab} + m)^2 - \left[(E_{lab})^2 - m^2\right] \\ &= 2mE_{lab} + 2m^2. \end{aligned} \tag{A.2.22}$$

In the center of mass frame, $p_{a,cm} = (E_{cm}/2, \vec{p}_{cm})$, $p_{b,cm} = (E_{cm}/2, -\vec{p}_{cm})$, and

$$s = |p_a + p_b|^2 = |p_{a,cm} + p_{b,cm}|^2 = (E_{cm})^2. \tag{A.2.23}$$

Setting these two expressions for s equal (since s is Lorentz invariant) gives,

$$E_{lab} = \frac{(E_{cm})^2}{2m} - m, \tag{A.2.24}$$

and in the ultra-relativistic limit $E \gg m$,

$$\sqrt{s} = E_{cm} \simeq \sqrt{2mE_{lab}}. \tag{A.2.25}$$

Since $\sqrt{s} = E_{cm}$, the total center of mass energy of a collision is often denoted by $\sqrt{s}$.

It is customary to take the momentum of a projectile as lying along the z-axis. The beam of projectiles is then parallel to the z-axis. The z-component of a particle is referred to as its *longitudinal momentum* $\vec{p}_L = (0, 0, p_z)$, and the *transverse momentum* $\vec{p}_T = (p_x, p_y, 0)$ is orthogonal to the beam axis. The magnitudes of the longitudinal and transverse momenta can be expressed as

$$p_L = p_z = |\vec{p}| \cos\theta, \quad p_T = \sqrt{p_x^2 + p_y^2} = |\vec{p}| \sin\theta, \tag{A.2.26}$$

where θ is the polar angle of the momentum vector with respect to the z-axis. Particles that are produced in a collision tend to have non-zero transverse momenta, whereas incident particles have zero or negligible transverse momenta. The *transverse mass* is defined as

$$m_T = \sqrt{p_T^2 + m^2}, \tag{A.2.27}$$

so that $E^2 = p_z^2 + m_T^2$.

One may consider the relativistic addition of velocities to define another convenient variable. Consider two velocities in one dimension with magnitudes β_1 and β_2. Velocities add as

$$\beta = \frac{\beta_1 + \beta_2}{1 + \beta_1\beta_2}, \tag{A.2.28}$$

where, working in natural units, $-1 < \beta < 1$. Now,

$$\tanh^{-1}\beta_1 + \tanh^{-1}\beta_2 = \tanh^{-1}\frac{\beta_1+\beta_2}{1+\beta_1\beta_2}, \tag{A.2.29}$$

and

$$\tanh^{-1}\beta = \frac{1}{2}\ln\frac{1+\beta}{1-\beta}, \tag{A.2.30}$$

so the *rapidity* y may be defined as

$$y \equiv \tanh^{-1}\beta = \frac{1}{2}\ln\frac{1+\beta}{1-\beta}, \tag{A.2.31}$$

and it is seen that the rapidity has the property of being additive, $y = y_1 + y_2$ for $\beta = (\beta_1+\beta_2)/(1+\beta_1\beta_2)$, where $y_1 = \tanh^{-1}\beta_1$ and $y_2 = \tanh^{-1}\beta_2$. Furthermore, a Lorentz transformation along the longitudinal axis, from one Lorentz frame S to another S', transforms the rapidity of a particle by $y' = y + \tanh^{-1}V$, where V is the velocity of S' with respect to S. For small velocities, $y \simeq \beta$, and for velocities approaching the speed of light, $y \to \pm\infty$ as $\beta \to \pm 1$. The properties of rapidity in relativistic mechanics are analogous to those of velocity in non-relativistic mechanics. Since $\beta = p_z/E$ in relativistic kinematics, the rapidity may also be expressed as

$$y = \frac{1}{2}\ln\frac{E+p_z}{E-p_z}. \tag{A.2.32}$$

A variable that is conveniently defined in terms of the polar angle of a particle's momentum is the *pseudorapidity*,

$$\eta \equiv -\ln\left(\tan\frac{\theta}{2}\right). \tag{A.2.33}$$

For high energies where particle masses are negligible so that $E^2 = |\vec{p}|^2 + m^2 \simeq |\vec{p}|^2$,

$$y \simeq \frac{1}{2}\ln\frac{|\vec{p}|+p_z}{|\vec{p}|-p_z} = \frac{1}{2}\ln\frac{1+\cos\theta}{1-\cos\theta} = -\ln\left(\tan\frac{\theta}{2}\right) = \eta, \tag{A.2.34}$$

and the rapidity and pseudorapidity are therefore equivalent for high energies where $E \gg m$.

The pseudorapidity may be expressed in terms of the momentum,

$$\eta = -\ln\left[\tan\left(\theta/2\right)\right] = \frac{1}{2}\ln\frac{|\vec{p}| + p_z}{|\vec{p}| - p_z}. \tag{A.2.35}$$

Then

$$e^{\eta} = \sqrt{\frac{|\vec{p}| + p_z}{|\vec{p}| - p_z}} \tag{A.2.36}$$

and

$$e^{-\eta} = \sqrt{\frac{|\vec{p}| - p_z}{|\vec{p}| + p_z}}, \tag{A.2.37}$$

so that

$$e^{\eta} + e^{-\eta} = 2\cosh\eta = 2|\vec{p}|/p_T, \tag{A.2.38}$$

and therefore

$$|\vec{p}| = p_T \cosh\eta. \tag{A.2.39}$$

Similarly,

$$e^{\eta} - e^{-\eta} = 2\sinh\eta = 2p_z/p_T, \tag{A.2.40}$$

and

$$p_z = p_T \sinh\eta. \tag{A.2.41}$$

Now, from these results, the pseudorapidity may be expressed in terms of the rapidity,

$$\eta = \frac{1}{2}\ln\frac{\sqrt{m_T^2\cosh^2 y - m^2} + m_T \sinh y}{\sqrt{m_T^2\cosh^2 y - m^2} - m_T \sinh y}. \tag{A.2.42}$$

A.3 Lorentz-Invariant Differential Cross Sections

Lorentz-invariant differential cross sections quantify particle production in relativistic collisions. Starting with the Lorentz transformation of a momentum four-vector[15],

$$\begin{aligned} E' &= \gamma\left(E - \beta p_z\right) \\ p'_x &= p_x \\ p'_y &= p_y \\ p'_z &= \gamma\left(p_z - \beta E\right), \end{aligned}$$

and from $E^2 - p_x^2 - p_y^2 - p_z^2 = m^2$, then for fixed p_x and p_y, it follows that $p_z dp_z = EdE$, and

$$\frac{dp'_z}{E'} = \frac{\gamma\left(dp_z - \beta dE\right)}{\gamma E\left(1 - \beta p_z/E\right)} = \frac{dp_z}{E}, \tag{A.3.43}$$

which is Lorentz-invariant. The Lorentz-invariant differential cross section for the production of a particle with momentum $\vec{p}$ in the phase-space element d^3p, and with energy E, is $Ed^3\sigma/d^3p$. Furthermore the phase-space element may be written as

$$d^3p = dp_x dp_y dp_z = d\phi p_T dp_T dp_z, \tag{A.3.44}$$

where p_T is the transverse momentum and ϕ is the azimuthal angle. Furthermore from $y = \frac{1}{2}\ln\frac{E+p_z}{E-p_z}$ for the rapidity, $dy = \frac{dp_z}{E}$ for fixed E, so the cross section may be written as

$$E\frac{d^3\sigma}{d^3p} = E\frac{d^3\sigma}{dp_x dp_y dp_z} = \frac{d^3\sigma}{p_T dp_T dy d\phi}. \tag{A.3.45}$$

The cross section can be expressed in terms of the variables p_T, y, or p_T, η, and these two forms are related by

$$\frac{1}{2\pi p_T}\frac{d^2\sigma}{d\eta dp_T} = \sqrt{1 - \frac{m^2}{m_T^2\cosh^2 y}}\,\frac{1}{2\pi p_T}\frac{d^2\sigma}{dy dp_T}. \tag{A.3.46}$$

The factor $\sqrt{1-\frac{m^2}{m_T^2\cosh^2 y}}$, appearing on the right-hand side of this equation, results from the transformation of the distribution from the rapidity variable to the pseudorapidity variable and is responsible for the dip at *mid-rapidity* ($\eta = 0$) that appears in the pseudorapidity distribution, whereas the rapidity distribution has a peak at $y = 0$. These distributions are identical at *forward rapidity* (large rapidity).

APPENDIX B: PYTHIA

PYTHIA is a Monte Carlo program that simulates high energy interactions[28]. Two incoming particles interact, resulting in a set of produced particles. The program provides an accurate representation of a wide range of particle interactions, including the strong interactions. In particular, hadronic interactions may be simulated, based on results and models from QCD. These physical processes are indeed very complex, especially when more than two particles are involved. In this study, PYTHIA is employed for simulating particle production in cold nucleus-nucleus collisions. A brief summary of PYTHIA is given in this chapter.

In a PYTHIA event, two incident hadrons approach each other, each characterized by a parton distribution. One parton from each hadron initiates a sequence of branchings, for example, $q \rightarrow qg$, resulting in an initial state shower. When these partons interact, many partons are generated if the process is soft, and usually two are generated if the process is hard. In addition, short-lived resonances may also be produced in hard processes. The outgoing partons then fragment into color neutral hadrons in a final state shower, and

unstable particles may decay further.

The PYTHIA program contains many switches and parameters for governing processes. Switches control the conditions under which processes occur, and the parameters determine the values of physical quantities. Most of these variables have default values, but other variables, such as the types of incoming particles, their collision energy, and number of collision events, must be specified by the user. Furthermore, the collision energy may be specified relative to the center of mass of the colliding particles or to a fixed frame where one particle is the target (at rest) and the other is the projectile.

A list of switches and parameters that are especially relevant to this study are listed in Table B.1. The first column gives the name of the switch or parameter, as implemented in the program, the second column gives the default value, and the third column specifies its purpose. The switch MSEL determines the type of processes that occur in an event. For MSEL = 0, the user is given full control of which subprocesses are to occur, and must specify them. This case is useful for studying specific processes and decays. In general, all types of collision events that may occur between the projectile and target, called *minimum bias* events, may be allowed. For MSEL = 1, inelastic, non-diffractive mininum bias events occur. For MSEL = 2, minimum bias events occur that also include elastic and diffractive events.

Table B.1: Important PYTHIA Switches.

Switch/Parameter	Default	Purpose
MSEL	1	Process Selection
MSTP(51)	7	Choice of PDF
PARP(91)	1.0	rms k_T in GeV/c

Switches that govern the structure of partonic interactions are designated by the letters "MSTP", and each of these switches is designated by a specific number, such as 51. The switch MSTP(51) specifies the parameterization of the parton distribution function (PDF). Each integer value for this switch corresponds to a particular parameterization, so that for example, the default value is 7, which corresponds to the CTEQ 5L (leading order) parameterization. There are a set of parton parameters labeled "PARP", each of which is also specified by a number. The PARP(91) parameter is the value of the root mean square (rms) transverse momentum, k_T, of the primordial partons, for a Gaussian k_T distribution. The primordial partons are those that constitute the incoming hadrons.

APPENDIX C: SIMULATION CODES

The C++ codes for the runpythia and glauber_mc programs are given in this appendix.

C.1 The runpythia Code

The computer code for the runpythia program is given here.

From the file runpythia.cc:

```
// runpythia.cc, Robert Zaballa
// For use with Pythia 6.225
//
// This program utilizes the ROOT TPythia6 class to interface with the
// fortran Pythia 6.225 event generator.
// Execution of this program requires  arguments.
// The first argument is the name of the OSCAR1999A format file to be
// used as input to PISA.
// The second argument is the name of the root output file containing a
// root tree that can be read by root or using the readpythia.cc program.
// The third argument is the number of events to simulate.
// The fourth argument is the collision energy (added by Robert Zaballa).
// The fifth argument is the rms k_T of the primordial partons.
// This is a modification of Chris Cleven's runpythia.cc .
```

```
#include <iostream>
#include <fstream>
#include <TObject.h>
#include <TPythia6.h>
#include <TMCParticle6.h>
#include <TFile.h>
#include <TTree.h>
#include <time.h>
#include "header.h"

#define C 299792458000 // mm/s from www.nist.gov

int main(int argc, char** argv)
{
  if (argc != 6) {
    std::cout << "Usage: runpythia [pisa output file][root output file] "
    << "[number of events]" "[C.M. Energy]" "[k_T]" << std::endl;
    return 1;
  }
  const char* pisaoutput = argv[1]; // name of Oscar format output file
  const char* rootoutput = argv[2]; // name of ROOT output file

  ofstream pisaout;

  TPythia6 pythia;     // create Pythia object

  int numberEvents = atoi(argv[3]);
  const double CMEnergy = atof(argv[4]);
  const double kTvalue = atof(argv[5]);
  //Proj. and Targ. Particles: p,pbar,pi+,pi-,pi0,n,nbar,gamma,e-,e+
  char targ[10] = "p"; //Target species
  char proj[10] = "p"; //Projectile species (p=proton, pbar=antiproton)
  float meankt = kTvalue; //Mean kT parameter (default value = 1.0 GeV/c)
  int partondist = 4;  //Parton distribution function
  float rootS = CMEnergy;   // Center of mass energy of system (in GeV)

  pythia.SetMSTP(51,partondist); // set PDF
  pythia.SetPARP(91,meankt);     // set mean kt
  pythia.SetMSEL(1);             // select process control:
                                 // (0 = user control)
                                 // (1 = Min Bias: pure inelastic)
 // (2 = Min Bias: diffractive
                                 //      & elastic collisions included)
```

```
/****THIS SECTION CONTAINS SEVERAL BLOCKS OF PROCESSES AND DECAYS****/
// codes for processes available in Pythia manual
// codes for decays are available from an internal table
// or in Pythia6225_decaytable.txt

/*
//Hard QCD Processes:   //f refers to any lepton or quark
pythia.SetMSUB(11,1);   //ff -> ff
pythia.SetMSUB(12,1);   //ffbar -> ffbar
pythia.SetMSUB(13,1);   //ffbar -> gg
pythia.SetMSUB(28,1);   //fg -> fg
pythia.SetMSUB(53,1);   //gg -> ffbar
pythia.SetMSUB(68,1);   //gg -> gg
*/

// Select processes to produce J/Psi
//pythia.SetMSUB(1,1);   // ffbar -> gamma* / Z0
//pythia.SetMSUB(86,1);  // gg -> J/Psi g
//pythia.SetMSUB(87,1);  // g+g -> chi0_c+g turned ON
//pythia.SetMSUB(88,1);  // g+g -> chi1_c+g turned ON
//pythia.SetMSUB(89,1);  // g+g -> chi2_c+g turned ON
//pythia.SetKFPR(86,1,100443); // force process to generate Upsilon(2S)
//pythia.SetMSUB(104,1); // g+g -> chi0_c
//pythia.SetMSUB(105,1); // g+g -> chi2_c
//pythia.SetMSUB(106,1); // gg -> J/Psi gamma
//pythia.SetKFPR(106,1,100443);// force process to generate Upsilon(2S)
//pythia.SetMSUB(107,1); // g gamma -> J/Psi g
//pythia.SetMSUB(108,1); // gamma gamma -> J/Psi gamma
//pythia.SetMSTP(43,1); //For Drell-Yan Process (gamma* only)

// Set decay channels for J/Psi
//pythia.SetMDME(858,1,0);  // J/Psi -> e+e-
//pythia.SetMDME(859,1,1);  // J/Psi -> mu+mu-
//pythia.SetMDME(860,1,0);  // J/Psi -> rndmflav
//pythia.SetMDME(1619,1,1); // chi_0c -> J/psi+gamma turned ON
//pythia.SetMDME(1620,1,0); // chi_0c -> rndmflav+rndmflavbar turned OFF
//pythia.SetMDME(1673,1,1); // chi_1c -> J/psi+gamma turned ON
//pythia.SetMDME(1674,1,0); // chi_1c -> rndmflav+rndmflavbar turned OFF
//pythia.SetMDME(979,1,1);  // chi_2c -> J/psi+gamma turned ON
//pythia.SetMDME(980,1,0);  // chi_2c -> rndmflav+rndmflavbar turned OFF
/*
for(int i = 174; i <= 189; i++) {
  pythia.SetMDME(i,1,0); //normal decay modes of Z0
}
```

```
pythia.SetMDME(180,1,-1);   //Non-standard decay modes of Z0
pythia.SetMDME(181,1,-1);   //""
pythia.SetMDME(188,1,-1);   //""
pythia.SetMDME(189,1,-1);   //""
pythia.SetMDME(184,1,1);    // Z0 -> mu+ mu-
pythia.SetMDME(182,1,1);    // Z0 -> e+ e-
*/
/*
// Set decay channels for Psi'
for (int i = 1567; i <= 1577; i++) {
 pythia.SetMDME(i,1,0);
}
pythia.SetMDME(1568,1,1); // turn on Psi' -> mu+mu-

// Set decay channels for Upsilon(1S)
for (int i = 1034; i <= 1042; i++) {
  pythia.SetMDME(i,1,0);
}
pythia.SetMDME(1035,1,1); // Upsilon -> mu+mu-

// Set decay channels for Upsilon(2S)
for (int i = 1578; i <= 1591; i++) {
  pythia.SetMDME(i,1,0);
}
pythia.SetMDME(1579,1,1); // Upsilon(2S) -> mu+mu-

// Set final-state radiation off
pythia.SetMSTP(71,0);

// Set only decay channel for Z0 to mu+mu-
for (int i = 174; i <= 189; i++) {
  pythia.SetMDME(i,1,0);  // normal decay modes of Z0
}
pythia.SetMDME(180,1,-1); // non-standard decay modes of Z0
pythia.SetMDME(181,1,-1); // "
pythia.SetMDME(188,1,-1); // "
pythia.SetMDME(189,1,-1); // "
pythia.SetMDME(184,1,1); // Z0 -> mu+ mu-
*/

// Initialize Pythia in center of mass system selected above
pythia.Initialize("CMS", proj, targ, rootS);
```

```
int nrandom = pythia.GetMRPY(1);
std::cout << " ******* " << std::endl;
std::cout << " *** Random from pythia: " << nrandom << std::endl;
std::cout << " ******* " << std::endl;

int rseedFromTime = time(NULL);

pythia.SetMRPY(1,rseedFromTime);
std::cout << " Time RSeed : " << rseedFromTime << std::endl;

nrandom = pythia.GetMRPY(1);
std::cout << " ******* " << std::endl;
std::cout << " *** Random from pythia: " << nrandom << std::endl;
std::cout << " ******* " << std::endl;


TClonesArray *particleArray;
TMCParticle *particle = new TMCParticle();

header *hdr = new header;

TClonesArray *part = new TClonesArray("TMCParticle", 500);

TFile *outFile = new TFile(rootoutput, "RECREATE",
                           "pythia TTree output");

TTree *pyttree = new TTree("pyttree", "pythia ttree output");
pyttree->Branch("event", "header", &hdr, 2048, 1);
pyttree->Branch("part", &part, 2048, 1);

// MAIN EVENT LOOP
for (int event = 1; event <= numberEvents; event++) {
  pythia.GenerateEvent();

  /*
  if (event <= 1000) {
    cout << "*************************" << endl;
    cout << "******event=" << event << endl;
    cout << "*************************" << endl;
  }
  */
  if (event == 1) {
    // Generate header of oscar output file
    pisaout.open(pisaoutput);
```

```
      pisaout << "# OSC1999A" << std::endl;
      pisaout << "# final_id_p_x" << std::endl;
      pisaout << "# runpythia" << std::endl;
      pisaout << "#" << std::endl;
    }

    // Get list of particles and global event information
    particleArray = (TClonesArray*)pythia.ImportParticles();
    int numParticles = particleArray->GetEntries();

    // Confirm these variables in the new PYTHIA manual
    hdr->SetEvt(event);          // Event number
    hdr->SetNpart(numParticles); // Total Number of particle entries
    hdr->SetProcessid(pythia.GetMSTI(1)); // Process ID
    hdr->SetX1(pythia.GetPARI(33));       // Bjorken x1, x2
    hdr->SetX2(pythia.GetPARI(34));
    hdr->SetSvar(pythia.GetPARI(14));     // partonic s,t,u
    hdr->SetTvar(pythia.GetPARI(15));
    hdr->SetUvar(pythia.GetPARI(16));
    hdr->SetQsqr(pythia.GetPARI(22));     // Q squared
    hdr->SetPt(pythia.GetPARI(17));       // transverse momentum

    pisaout << 0 << " " << numParticles-2 << std::endl;

    int nstore = 0;
    for (int i = 0; i < numParticles; i++) {
      particle = (TMCParticle*)particleArray->At(i); // get particle info

      // print particle information in oscar file
      if (nstore >= 2) {
pisaout << nstore - 2 << " "             // particle number
<< particle->GetKF() << " "      // PDG particle id
<< 0 << " "                      // additional particle id info
<< particle->GetPx() << " "      // particle 4-momentum
<< particle->GetPy() << " "
<< particle->GetPz() << " "
<< particle->GetEnergy() << " "
<< particle->GetMass() << " "    // particle mass
<< particle->GetVx()*1.e12 << " "// particle vertex in mm
<< particle->GetVy()*1.e12 << " "
<< particle->GetVz()*1.e12 << " "
<< particle->GetTime() / C       // creation time
<< std::endl;
      }
```

```
      new ((*part)[nstore++]) TMCParticle(*particle);
    }

    pisaout << 0 << " " << 0 << std::endl;

    pyttree->Fill();
    part->Clear();
    particleArray->Clear();

    if (event%1000 == 0)
    std::cout << event << " events processed" << std::endl;
  }

  pyttree->Write();
  outFile->Close();
  pisaout.close();

  std::cout << "Finished" << std::endl;
  return 0;
}
```

From the file header.h:

```
// ================================================
//
// Make these objects of type TObject and make a set of
// Get and Put methods for all private data members
// Is this the best way?

//
// Created 06-07-2001 James Nagle
// make sure to add variable definitions here !
// ================================================

// by making this an object of type TObject I can use
// TClonesArray and then easily store things in a TTree
// then when you read the TTree you just need to include
// this header file and you can access the data by these
// get methods
//
// add ifdef-ing on this routine

#ifndef ROOT_TObject
#include <TObject.h>
```

```
#endif

class header : public TObject {

private:

  Int_t   evt;
  Int_t   npart;
  Int_t   processid;
  Float_t x1;
  Float_t x2;
  Float_t svar;
  Float_t tvar;
  Float_t uvar;
  Float_t qsqr;
  Float_t pt;

public:

  header(){;}              // constructor
  virtual ~header(){;}     // destructor

  Int_t   GetEvt()        {return evt;};   // just inline the get here
  Int_t   GetNpart()      {return npart;};
  Int_t   GetProcessid()  {return processid;};
  Float_t GetX1()         {return x1;};
  Float_t GetX2()         {return x2;};
  Float_t GetSvar()       {return svar;};
  Float_t GetTvar()       {return tvar;};
  Float_t GetUvar()       {return uvar;};
  Float_t GetQsqr()       {return qsqr;};
  Float_t GetPt()         {return pt;};

  void    SetEvt(Int_t Evt)               {evt = Evt;};
  void    SetNpart(Int_t Npart)           {npart = Npart;};
  void    SetProcessid(Int_t Processid)   {processid = Processid;};
  void    SetX1(Float_t X1)               {x1 = X1;};
  void    SetX2(Float_t X2)               {x2 = X2;};
  void    SetSvar(Float_t Svar)           {svar = Svar;};
  void    SetTvar(Float_t Tvar)           {tvar = Tvar;};
  void    SetUvar(Float_t Uvar)           {uvar = Uvar;};
  void    SetQsqr(Float_t Qsqr)           {qsqr = Qsqr;};
  void    SetPt(Float_t Pt)               {pt = Pt;};
```

```
  ClassDef(header,2)

};
```

From the file header.C:

```
#include "header.h"

// this is a macro
ClassImp(header)
```

C.2 The glauber_mc Code

The computer code for the glauber_mc program is given here.

From the file glauber_mc.cc:

```
/*!
mainpage PHENIX Glauber Monte Carlo Program

This program determines the number of participants (Npart)
and the number of binary nucleon-nucleon collisions (Ncoll)
in heavy ion reactions.

The root output file contains a ntuple with all necessary information
to extract Npart and Ncoll for given centrality cuts.

Klaus Reygers,
University of Muenster

Modifications made by Robert Zaballa and Xiaochun He,
Georgia State University.
last updated: March 4, 2008
*/

#include <iostream>
#include <fstream>
#include <cstring>
#include <cctype>
#include <string>
#include <math.h>
```

```
#include <stdio.h>
#include <time.h>

#include "TROOT.h"
#include "TF1.h"
#include "TH1.h"
#include "TH2.h"
#include "TNtuple.h"
#include "TFile.h"
#include "TRandom.h"

//Added 3/13/07:
#include "PythiaEventAccessor.h"
//Added 3/19/07:
#include <TObject.h>
#include <TPythia6.h>
#include <TMCParticle6.h>
#include <TTree.h>
#include <TH1I.h>
#include <TH1F.h>
#include "TBranch.h"
#include "TClonesArray.h"
#include "headerAA.h"

using namespace std;

//! functions for reading glauber settings input file
void GetNumValue(ifstream&, double&);
void GetStringValue(ifstream&, string&);

TROOT root("Glauber","Glauber Npart calculation");

const double pi = 3.14159265358979323846264643;
const double m0 = 0.93827;  //Rest mass of proton in GeV/c^2
const double m02 = m0*m0;   //m0^2
const double m04 = m02*m02; //m0^4

enum nucleonType {neutron, proton};

PythiaEventAccessor *p;

PythiaEventAccessor *p10 = new PythiaEventAccessor(10);
PythiaEventAccessor *p15 = new PythiaEventAccessor(15);
PythiaEventAccessor *p20 = new PythiaEventAccessor(20);
```

```
PythiaEventAccessor *p25 = new PythiaEventAccessor(25);
PythiaEventAccessor *p30 = new PythiaEventAccessor(30);
PythiaEventAccessor *p35 = new PythiaEventAccessor(35);
PythiaEventAccessor *p40 = new PythiaEventAccessor(40);
PythiaEventAccessor *p45 = new PythiaEventAccessor(45);
PythiaEventAccessor *p50 = new PythiaEventAccessor(50);
PythiaEventAccessor *p55 = new PythiaEventAccessor(55);
PythiaEventAccessor *p60 = new PythiaEventAccessor(60);
PythiaEventAccessor *p65 = new PythiaEventAccessor(65);
PythiaEventAccessor *p70 = new PythiaEventAccessor(70);
PythiaEventAccessor *p75 = new PythiaEventAccessor(75);
PythiaEventAccessor *p80 = new PythiaEventAccessor(80);
PythiaEventAccessor *p85 = new PythiaEventAccessor(85);
PythiaEventAccessor *p90 = new PythiaEventAccessor(90);
PythiaEventAccessor *p95 = new PythiaEventAccessor(95);
PythiaEventAccessor *p100 = new PythiaEventAccessor(100);
PythiaEventAccessor *p105 = new PythiaEventAccessor(105);
PythiaEventAccessor *p110 = new PythiaEventAccessor(110);
PythiaEventAccessor *p115 = new PythiaEventAccessor(115);
PythiaEventAccessor *p120 = new PythiaEventAccessor(120);
PythiaEventAccessor *p125 = new PythiaEventAccessor(125);
PythiaEventAccessor *p130 = new PythiaEventAccessor(130);
PythiaEventAccessor *p135 = new PythiaEventAccessor(135);
PythiaEventAccessor *p140 = new PythiaEventAccessor(140);
PythiaEventAccessor *p145 = new PythiaEventAccessor(145);
PythiaEventAccessor *p150 = new PythiaEventAccessor(150);
PythiaEventAccessor *p155 = new PythiaEventAccessor(155);
PythiaEventAccessor *p160 = new PythiaEventAccessor(160);
PythiaEventAccessor *p165 = new PythiaEventAccessor(165);
PythiaEventAccessor *p170 = new PythiaEventAccessor(170);
PythiaEventAccessor *p175 = new PythiaEventAccessor(175);
PythiaEventAccessor *p180 = new PythiaEventAccessor(180);
PythiaEventAccessor *p185 = new PythiaEventAccessor(185);
PythiaEventAccessor *p190 = new PythiaEventAccessor(190);
PythiaEventAccessor *p195 = new PythiaEventAccessor(195);
PythiaEventAccessor *p200 = new PythiaEventAccessor(200);

vector<particleStore> partVec;

// Create a ROOT Tree for produced particles
TTree *GLtree = new TTree("GLtree","Particle Tree");

TMCParticle *particle = new TMCParticle();
```

```
headerAA *hdrAA = new headerAA;
TClonesArray *part = new TClonesArray("TMCParticle",500);

int nAAColl;

//! random numbers from negative binomial distribution
class ranNBD {
  double mu, k;    // parameters for negative binomial distr.
  TH1D   hNBD;     // histogram for negative binomial distr.
  int    nNBDBins; // number of bins for hNBD

  void init();

public:
  ranNBD(double, double);
  void   SetParameters(double, double);
  double getValue(int);
  int    getRandom();
};

//! Class to store nucleon properties
class nucleon {
public:
  double      x,y,z;  // spatial coordinates
  nucleonType type;   // proton or neutron
  int         ncoll;     // number of nn-collision it has suffered
  int         ncollHard; // also count coll. with very small x-section
  // (used as a counter for collisions which create a J/Psi
};

//! Class to store properties of the nuclei
class nucleus {
  int       A;                // mass number
  int       Z;                // proton number
  double    x,y,z;            // spatial coordinates of the center
  double    ws_radius;        // Woods-Saxon radius parameter
  double    ws_diff;          // Woods-Saxon diffuseness
  const static int MaxNucleons = 250;
  nucleon   nucl[MaxNucleons]; // array of nucleons
  nucleon*  part[MaxNucleons]; // array with pointers to the participants

  // auxillary stuff
  TF1*    ws;  // pointer to Woods-Saxon Distribution
  TF1*    th;  // pointer to sin-distribution
```

```
  TF1*    fDeutDist;  // for the deuteron: prob. distribution for
  // distance between the proton and the neutron

public:
  nucleus(int, int);   // Woods-Saxon param. from parameterization
  nucleus(int, int, const double&, const double&);
  ~nucleus();
  int  GetMass();
  int  GetProtons();
  void SetCoordinates(double, double, double);
  void DistributeNucleons();
  void DistributeHardCoreNucleons();  // alternative to DistributeNucleons
  void ResetNucleonCollCounter();

  friend class AACollision;

private:
  void DefineShape();
  void InitNucleonPositions();
};


class jpsi_coll {
public:
  int ip; // nucleon number in projectile
  int it; // nucleon number in target
  double x;
  double y;
};

//! Class that simulates collisions of two nuclei
class AACollision {
  nucleus* Projectile;
  nucleus* Target;
  nucleon* Part[500];    // pointers to the participants
  double sigma_nn_inel_fm2;  // inel. nucleon-nucleon cross section in fm^2
  double sigma_nn_tot_fm2;   // total nucleon-nucleon cross section in fm^2
  double sigma_nn_hard_fm2;  // x-section for a hard process
  double sigma_jpsi_abs_fm2; // J/Psi absorption cross section in normal
                             // nuclear matter
  double    ecms;            // center of mass energy in GeV
  double    Eloss;           // Energy Loss Fraction
  double    b;
  int       NpartProj;
```

```
  int       NpartTarg;
  int       Npart;
  int       Ncoll;
  int       NcollHard;       // counter for collisions which create a J/Psi
  int       NcollHardSurv;   // counter for number of surviving J/Psi's
  int       NSumSubsColl;    // needed for calculation of mean path length
                             // of J/Psi in normal nuclear matter
  double    Eccentricity;    // standard eccentricity
  double    EccentricityWERCS;  // ecc. Without External Reference
                                // Coordinate System
  double    v2Part;             // v2 of the participant distribution

  double    OverlapArea;
  double    alpha;              // Nhit,BBC ~ N_part ^ alpha

  // parameters for Gaussian overlap function
  double    fGausOvlP1;
  double    fGausOvlP2;

  // switches the specify details of the simulation
  bool HardCoreNucleons;
  bool ZdcCheck;
  int  NNOverlapFunc;

  ranNBD nbd;

public:
  AACollision();
  AACollision(nucleus*, nucleus*);
  void   SetInelNNCrossSection(const double&);
  void   SetTotalNNCrossSection(const double&);
  void   SetHardNNCrossSection(const double&);
  void   SetJPsiAbsCrossSection(const double&);
  void   SetPhenixBBCNbdParams(const double&, const double& k);
  void   SetPhenixBBCMultScalingPower(const double&);
  void   SetCMSEnergy(double );
  void   SetELossFraction(double ); //RZ. added 6/21/07
  int    Collide(int);
  void   SetImpactParameter(const double&);
  double GetImpactParameter();
  double GetImpactParTargPart(int);
  double GetImpactParProjPart(int);
  void   GetNpartNcoll(int&, int&, int&, int&, int&, int&);
  void   GetPhenixZDCEnergy(double&, double&, double&, double&, int&);
```

```
  int    GetNhitsFromNBD();

  double GetEccentricity();
  double GetEccentricityWERCS();
  double GetPartV2();
  double GetOverlapArea();

  void   InitGausOvlPara();

  //3/12/07: CMVelocity, CMEnergy, and CMRapidityShift added by R. Zaballa
  double cmV;
  double cmE;
  double rap; //Rapidity variable (can't call it 'y', 'y' is used already)
  double dy;
  double Rapidity(double, double);
  void NNCMVelocity(double, int, int);   //CM Velocity for each collision
  void NNCMEnergy(double, int, int);  //Total CME of nucleons before a coll
  double NNRapidityShift(double);   //Rapidity shift for collision CM


  // switches
  void SetHardCoreNucleons(bool);
  void SetZdcCheck(bool);
  void SetNNOverlapFunc(int);

private:
  void CalcEccentricityAndArea();
  bool NNCollision (const double&);
};


//! random numbers from negative binomial distribution
ranNBD::ranNBD(double nbd_mu = 4.04, double nbd_k = 2.54) {

  mu = nbd_mu;
  k  = nbd_k;

  nNBDBins = 101;
  hNBD = TH1D("hNBD","NBD histogram",nNBDBins,-0.5,nNBDBins-0.5);

  init();
}
```

```
//! fill NBD histogram
void ranNBD::init() {
  hNBD.Reset();
  for(int i=0; i<nNBDBins; i++) {
    hNBD.SetBinContent(i+1,getValue(i));
  }
}

//! Set parameters for negative binomial distr.
void ranNBD::SetParameters(double nbd_mu, double nbd_k) {
  mu = nbd_mu;
  k  = nbd_k;

  init();
}

//! Return the value of the negative binomial distribution
//! with parameters (mu, k) and for given n.
//! Function taken from Sasha Milov.
double ranNBD::getValue(int n) {

  double F;
  double f;

  F  = TMath::Gamma(n + k) / TMath::Gamma(n + 1.);
  F /= TMath::Gamma(k);
  f  = n * TMath::Log(mu/k) - (n + k) * TMath::Log(1.0 + mu/k);
  f  = TMath::Exp(f);
  F *= f;

  return F;
}

//! sample negative binomial distribution
int ranNBD::getRandom() {
  return int(hNBD.GetRandom() + 0.499999999);
}


//! Creates a nucleus. Woods-Saxon parameters are taken
//! from parameterization.
nucleus::nucleus(int a_nucleus, int z_nucleus) {

  A = a_nucleus;
```

```
  Z = z_nucleus;

  // Woods-Saxon parameters
  switch (A) {
    case 197: ws_radius = 6.38;
              ws_diff   = 0.54;
              break;

    case  63: ws_radius = 4.2064;
              ws_diff   = 0.5977;
              break;

    default:
       // new parameterization
       // (from textbook: Povh, Rith, Scholz, Zetsche:
       //  Teilchen und Kerne)
              ws_radius = 1.07*pow(A,1./3.);
              ws_diff   = 0.546;
              break;
  }

  InitNucleonPositions();
  DefineShape();
}


//! Creates a nucleus. Woods-Saxon parameters are explicitely
//! set by the user
nucleus::nucleus(int a_nucleus, int z_nucleus,
                 const double& wsr, const double& wsd) {
  A = a_nucleus;
  Z = z_nucleus;

  // Woods-Saxon parameters
  ws_radius = wsr;
  ws_diff   = wsd;

  InitNucleonPositions();
  DefineShape();

}

//! Deallocates used memory
nucleus::~nucleus() {
```

```
  delete ws;
  delete th;
  delete fDeutDist;
}

//! Defines Woods-Saxon distribution
void nucleus::DefineShape() {

  // probability distribution of radius
  ws = new TF1("f1","x**2/(1+exp((x-[0])/[1]))",0.,20.);
  ws->SetParameters(ws_radius,ws_diff);

  // probability distribution of theta
  th = new TF1("f2","sin(x)",0.,pi);

  // For the deuteron: probability distribution for the distance
  // between the proton and the neutron.
  // This is the square of the Hulthen wave function, see
  // Hodgson, Nuclear Reactions and Nuclear Structure, p.453,
  // where the parameters are taken from.
  // double fHulPar1 = 0.23; // 1/fm
  // double fHulPar2 = 1.61; // 1/fm

  // try Hulthen parameters from Dave's analysis note (AN165)
  double fHulPar1 = 0.228;  // 1/fm
  double fHulPar2 = 1.18;   // 1/fm

  fDeutDist = new TF1("f3",
    "x^2*([0]*[1]*([0]+[1]))/(2*pi*(pow([0]-[1],2)))"
    "*pow((exp(-[0]*x)-exp(-[1]*x))/x,2)",0.,20.);

  fDeutDist->SetParameters(fHulPar1, fHulPar2);

}

//! Resets spatial coordinates of all nucleons of a nucleus
//! to zero
void nucleus::InitNucleonPositions() {

  x = 0.;
  y = 0.;
  z = 0.;
  for(int i=0; i<A; i++) {
    nucl[i].x     = 0.;
```

```
    nucl[i].y     = 0.;
    nucl[i].z     = 0.;
    nucl[i].type  = neutron;
    nucl[i].ncoll = 0;
  }

}

//! Return mass number of the nucleus
int nucleus::GetMass() {
  return A;
}

//! Return number of protons
int nucleus::GetProtons() {
  return Z;
}

//! Sets coordinates of the center of the nucleus
void nucleus::SetCoordinates(double xc, double yc, double zc) {
  x = xc;
  y = yc;
  z = zc;
}

//! Randomly distributes nucleons according to a Woods-Saxon nuclear
//! density distribution
void nucleus::DistributeNucleons() {
  double r, theta, phi, xc, yc, zc;

  if (A>2) {

    // loop over all nucleons
    for(int i=0; i<A; i++) {

      r     = ws->GetRandom();
      theta = th->GetRandom();
      phi   = 2.* pi * gRandom->Rndm();

      // coordinates in local coordinate system
      xc = r * sin(theta) * cos(phi);
      yc = r * sin(theta) * sin(phi);
      zc = r * cos(theta);
```

```
    // translate coordinates to global coordinate system
    nucl[i].x = x + xc;
    nucl[i].y = y + yc;
    nucl[i].z = z + zc;
  }
}

else if (A==2) {

  // we have a deuteron

  double fProtonNeutronDist = fDeutDist->GetRandom();

  // the first nucleon
  double r1     = fProtonNeutronDist / 2.;
  double theta1 = th->GetRandom();
  double phi1   = 2.* pi * gRandom->Rndm();


  // the second nucleon
  double r2     = r1;
  double theta2 = pi - theta1;
  double phi2   = phi1 + pi;
  if (phi2 >= 2*pi) phi2 -= 2*pi;

  // coordinates in global coordinate system
  nucl[0].x = x + (r1 * sin(theta1) * cos(phi1));
  nucl[0].y = y + (r1 * sin(theta1) * sin(phi1));
  nucl[0].z = z + (r1 * cos(theta1));

  nucl[1].x = x + (r2 * sin(theta2) * cos(phi2));
  nucl[1].y = y + (r2 * sin(theta2) * sin(phi2));
  nucl[1].z = z + (r2 * cos(theta2));

}

else {

  // don't distribute a single proton or neutron
  nucl[0].x = x;
  nucl[0].y = y;
  nucl[0].z = z;

}
```

```
  // since nucleons are distributed randomly we can define
  // the first Z nucleons as protons, the rest as neutrons
  for (int i=0; i<Z; i++) nucl[i].type = proton;

}

//! Same as nucleus::DistributeNucleons() but nucleons are
//! required to have a minimum distance of 0.8 fm, i.e.
//! a nucleon hard core of 0.4 fm is assumed.
void nucleus::DistributeHardCoreNucleons() {

  // take hard core of nucleons into account,
  // this method can be calles instead of DistributeNucleons()

  // hard core radius of nucleons (fm)
  const double hcr = 0.4;

  double r, theta, phi, xc, yc, zc;

  int verbose = 0;
  if (verbose > 10) {
    cout << endl;
    cout << "Entering nucleus::DistributeHardCoreNucleons !" << endl;
    cout << "-----------------------------------------------" << endl;
  }

  // check if we really have a nucleus and not just a single
  // proton or neutron or a deuteron
  if (A > 2) {

    int iNucleusTry = 0;   // number of attempted nucleus creations
    int nd = 0;     // number of already distributed nucleons
    do {

      r = ws->GetRandom();

      int Overlap;
      int iNucleonTry = 0;
      do {

        iNucleonTry++;
        Overlap = 0;
```

```
      theta = th->GetRandom();
      phi   = 2.* pi * gRandom->Rndm();

      xc = r * sin(theta) * cos(phi);
      yc = r * sin(theta) * sin(phi);
      zc = r * cos(theta);

      nucl[nd].x = xc;
      nucl[nd].y = yc;
      nucl[nd].z = zc;

      for (int i=0; i<nd; i++) {
        double d = sqrt(pow(nucl[i].x - xc,2) + pow(nucl[i].y - yc,2) +
                        pow(nucl[i].z - zc,2));
        if (d < 2*hcr) {
          Overlap = 1;
          break;
        }
      }

    } while (Overlap == 1 && iNucleonTry<10000);

    // If the last nucleon couldn't be distributed,
    // start assembling the whole nucleus again.
    if (Overlap == 1) {
      nd = 0;
      iNucleusTry++;
    }
    else {
      nd++;
      if (verbose > 10)
        cout << "Nucleon " << nd << ": " << iNucleonTry << " try/tries"
             << ", r = " << r << endl;
    }
  } while (nd < A  && iNucleusTry<10000);

  if (nd < A) {
  cout << "nucleus::DistributeHardCoreNucleons: Couldn't Make Nucleus!"
       << endl;
  exit(-1);
  }
  else {
    for (int i=0; i<A; i++) {
// add coordinated of center
```

```
      nucl[i].x += x;
      nucl[i].y += y;
      nucl[i].z += z;
    }

    // since nucleons are distributed randomly we can define
    // the first Z nucleons as protons, the rest as neutrons
    for (int i=0; i<Z; i++) nucl[i].type = proton;
  }
}

else if (A == 2) {

  // we have a deuteron

  double fProtonNeutronDist;
  int   iDeuteronTry = 0;
  int   iDeuteronTryMax = 10000;

  do {
    iDeuteronTry++;
    fProtonNeutronDist = fDeutDist->GetRandom();
  } while (fProtonNeutronDist < 2*hcr && iDeuteronTry < iDeuteronTryMax);

  if (iDeuteronTry >= iDeuteronTryMax) {
  cout << "nucleus::DistributeHardCoreNucleons: Couldn't Make Deuteron!"
       << endl;
  exit(-1);
  }
  else {

    // the first nucleon
    double r1     = fProtonNeutronDist / 2.;
    double theta1 = th->GetRandom();
    double phi1   = 2.* pi * gRandom->Rndm();

    // the second nucleon
    double r2     = r1;
    double theta2 = pi - theta1;
    double phi2   = phi1 + pi;
    if (phi2 >= 2*pi) phi2 -= 2*pi;

    // coordinates in global coordinate system
    nucl[0].x = x + (r1 * sin(theta1) * cos(phi1));
```

```
      nucl[0].y = y + (r1 * sin(theta1) * sin(phi1));
      nucl[0].z = z + (r1 * cos(theta1));

      nucl[1].x = x + (r2 * sin(theta2) * cos(phi2));
      nucl[1].y = y + (r2 * sin(theta2) * sin(phi2));
      nucl[1].z = z + (r2 * cos(theta2));

    }
  }

  else {

    // don't distribute a single proton or neutron
    nucl[0].x = x;
    nucl[0].y = y;
    nucl[0].z = z;

    nucl[0].type = proton;
    if (Z == 0) nucl[0].type = neutron;

  }

}

//! Reset the counter for the number of nn-collisions for each
//! nucleon to zero
void nucleus::ResetNucleonCollCounter() {
  for (int i=0; i<A; i++) nucl[i].ncoll = 0;
}

//! Creates an object that simulates a AA collision
AACollision::AACollision(nucleus* A,nucleus* B) {
  Projectile = A;
  Target     = B;

  // default values for switches
  HardCoreNucleons = false;
  ZdcCheck         = false;

  // default nucleon-nucleon overlap function = black disk
  NNOverlapFunc = 0;

}
```

```
//! Switch that controls if nucleon hard core is taken in to account
void AACollision::SetHardCoreNucleons(bool b) {
  HardCoreNucleons = b;
}

//! Switch that controls the neutron loss function used
//! in the ZDC simulation.
/*!
The default simulation uses the neutron loss function that
gave the best agreement with measured ZDC spectra.
If the check is activated a worse neutron loss function
is used. The neutron loss function describes which fraction
of spectator neutrons is not seen in the ZDC due to
coalescence (formation of deuterons and heavier charged fragments).
*/
void AACollision::SetZdcCheck(bool b) {
  ZdcCheck = b;
}

//! Sets the used nucleon-nucelon overlap function.
//! 0 = black disk, 1 = gray disk, 2 = gaussian
void AACollision::SetNNOverlapFunc(int id) {
  NNOverlapFunc = id;
}

//! Sets inelastic nucleon-nucleon collision cross section
void AACollision::SetInelNNCrossSection(const double& sigma_inel_mb) {

  // input: mb
  // internally used: fm^2

  // translate mb to fm^2
  sigma_nn_inel_fm2 = sigma_inel_mb / 10.;
}

//! Sets total nucleon-nucleon collision cross section
void AACollision::SetTotalNNCrossSection(const double& sigma_tot_mb) {

  // input: mb
  // internally used: fm^2

  // translate mb to fm^2
  sigma_nn_tot_fm2 = sigma_tot_mb / 10.;
}
```

```
//! Sets a cross section for the occurence of hard processes
/*!
The feature was only used as a check to see whether the ratio
Ncoll/sigma somehow depends on the cross section sigma.
As expected, Ncoll/sigma was found to be independent of
sigma.
*/
void AACollision::SetHardNNCrossSection(const double& sigma_hard_mb) {

  // input: mb
  // internally used: fm^2

  // translate mb to fm^2
  sigma_nn_hard_fm2 = sigma_hard_mb / 10.;
}

//! Sets a cross section for the
void AACollision::SetJPsiAbsCrossSection(const double& sigma_jpsi_abs) {

  // input: mb
  // internally used: fm^2

  // translate mb to fm^2
  sigma_jpsi_abs_fm2 = sigma_jpsi_abs / 10.;
}


//! Set center of mass energy per nn-pair of the AA collision.
/*!
The center of mass energy is needed in the simulation of
the ZDC response. It is also used to determine the average
charged multiplicity in the BBC acceptance.
*/
void AACollision::SetCMSEnergy(double e) {
  // center of mass energy in GEV
  ecms = e;
}

void AACollision::SetELossFraction(double ELF) {
  // Energy Loss fraction for NN Collisions (RZ, added 6/21/07):
  Eloss = ELF;
}
```

```
//! set parameters for negative binomial distribution used
//! in the simulation of the BBC signal
void AACollision::SetPhenixBBCNbdParams(const double& mu, const double& k)
{
  nbd.SetParameters(mu, k); // Au+Au at 200 GeV
}


//! set power alpha that determines scaling of BBC hit mult. with N_part
void AACollision::SetPhenixBBCMultScalingPower(const double& a) {
  alpha = a;
int NNEvents = 0;
int AAEvents = 0;
}

//! This function decides whether a nucleon-nucleon collision
//! takes place or not.
/*! Three different nn-overlap function are implemented:
    black disk, gray disk and a gaussian overlap function.
    The overlap function gives, as a function of the nucleon-
    nucleon impact parameter, the probability that a nucleon-
    nucleon collision takes place. It is normalized such that
    the area integral over the overlap function gives the total
    inelastic nucleon-nucleon cross section.
 */
bool AACollision::NNCollision(const double& dxy2) {

  switch(NNOverlapFunc) {

  case 0: {   // black disk
    return (dxy2 < sigma_nn_inel_fm2/pi);
  }

  case 1: {   // gray disk
    if (dxy2 < sigma_nn_tot_fm2/pi) {
      double prob = sigma_nn_inel_fm2/sigma_nn_tot_fm2;
      if (gRandom->Rndm() <= prob) return true;
      else return false;
    }
    else return false;
  }

  case 2: {   // gaussian overlap function
    double prob = 1.-pow(1-fGausOvlP1*exp(-fGausOvlP2*dxy2),2);
```

```
    if (gRandom->Rndm() <= prob) return true;
    else return false;
  }

  default:
    cout << "AACollision::NNCollision:
             overlap function not specified" << endl;
    exit(-1);

  }
};

//3/12/07: Added by Robert Zaballa:
void AACollision::NNCMVelocity(double Eloss, int projHits, int targHits){

  double ecms2 = ecms*ecms;

  cmV = (sqrt(ecms2*0.25*pow((1-Eloss), 2*projHits) - m02)
    - sqrt(ecms2*0.25*pow((1-Eloss),2*targHits) - m02))
              /(ecms*0.5*(pow((1-Eloss), projHits)
+ pow((1-Eloss), targHits)));
};

//3/12/07: Added by Robert Zaballa:
void AACollision::NNCMEnergy(double Eloss, int projHits, int targHits){

  double ecms2 = ecms*ecms;
  double ecms4 = ecms2*ecms2;

  if((pow((1-Eloss),projHits) >= m0/(.5*ecms))
     && (pow((1-Eloss),targHits) >= m0/(.5*ecms))) {

    cmE = sqrt(2*(m02 + ecms2*0.25*pow((1-Eloss),(projHits + targHits)))
     + 2*sqrt(ecms4*0.0625*pow((1-Eloss),2*(projHits + targHits))
      - m02*ecms2*0.25*(pow((1-Eloss), 2*projHits)
      + pow((1-Eloss),2*targHits)) + m04));
  }
  else cmE = -1;
};

//3/12/07: Added by Robert Zaballa:
double AACollision::NNRapidityShift(double cmV) {
  dy = 0.5*log((1 + cmV)/(1 - cmV));
```

```
  return dy;
};

double AACollision::Rapidity(double E, double pz) {
  rap = 0.5*log((E + pz)/(E - pz));
  return rap;
};

//! This function simulates the collision of two nuclei
//! and counts Npart and Ncoll.
int AACollision::Collide(int nAAcoll) {

  double dxy2;

  // reset counters
  NpartProj     = 0;
  NpartTarg     = 0;
  Npart         = 0;
  Ncoll         = 0;
  NcollHard     = 0;
  NcollHardSurv = 0;
  NSumSubsColl  = 0;

  // array for collisions which create a J/Psi for later evaluation of
  // J/Psi survival probability
  const int NcollHardMax = 10000;
  jpsi_coll jpc[NcollHardMax];

  // Reset counters of the two nuclei
  Target->ResetNucleonCollCounter();
  Projectile->ResetNucleonCollCounter();


  if(!HardCoreNucleons) {
    Target->DistributeNucleons();
    Projectile->DistributeNucleons();
  }
  else {
    Target->DistributeHardCoreNucleons();
    Projectile->DistributeHardCoreNucleons();
  }


  for (int ip=0; ip<Projectile->GetMass(); ip++) {
```

```
    for (int it=0; it<Target->GetMass(); it++) {

      // squared transverse distance of nucleons
      dxy2 = pow((Projectile->nucl[ip].x - Target->nucl[it].x),2) +
             pow((Projectile->nucl[ip].y - Target->nucl[it].y),2);

      // check if there is a nn collision
      if (NNCollision(dxy2)) {
//Ncoll++; //(Original placement of Ncoll++)

// needed for average J/Psi path length in nuclear matter
// now done below usin alternatic formula
// NSumSubsColl += Projectile->nucl[ip].ncoll;
// NSumSubsColl += Target->nucl[it].ncoll;

NNCMEnergy(Eloss, Projectile->nucl[ip].ncoll, Target->nucl[it].ncoll);
NNCMVelocity(Eloss, Projectile->nucl[ip].ncoll, Target->nucl[it].ncoll);
/* //Diagnostic Check:
cout << "ip: " << ip << " ProjHits: " << Projectile->nucl[ip].ncoll
     << " it: " << it << " TargHits: " << Target->nucl[it].ncoll
     << " TargEnergy: " << 100*pow(Eloss,Target->nucl[it].ncoll)
     << " Total NN CME: " << cmE
     << " Ncoll: " << Ncoll << endl;
*/

cmE = (int)(cmE+0.5); //round cmE to nearest integer
if(cmE < 5) break;
Ncoll++;
if((cmE >= 5) && (cmE < 13)) p = p10;
if((cmE >= 13) && (cmE < 18)) p = p15;
if((cmE >= 18) && (cmE < 23)) p = p20;
if((cmE >= 23) && (cmE < 28)) p = p25;
if((cmE >= 28) && (cmE < 33)) p = p30;
if((cmE >= 33) && (cmE < 38)) p = p35;
if((cmE >= 38) && (cmE < 43)) p = p40;
if((cmE >= 43) && (cmE < 48)) p = p45;
if((cmE >= 48) && (cmE < 53)) p = p50;
if((cmE >= 53) && (cmE < 58)) p = p55;
if((cmE >= 58) && (cmE < 63)) p = p60;
if((cmE >= 63) && (cmE < 68)) p = p65;
if((cmE >= 68) && (cmE < 73)) p = p70;
if((cmE >= 73) && (cmE < 78)) p = p75;
if((cmE >= 78) && (cmE < 83)) p = p80;
if((cmE >= 83) && (cmE < 88)) p = p85;
```

```
if((cmE >= 88) && (cmE < 93)) p = p90;
if((cmE >= 93) && (cmE < 98)) p = p95;
if((cmE >= 98) && (cmE < 103)) p = p100;
if((cmE >= 103) && (cmE < 108)) p = p105;
if((cmE >= 108) && (cmE < 113)) p = p110;
if((cmE >= 113) && (cmE < 118)) p = p115;
if((cmE >= 118) && (cmE < 123)) p = p120;
if((cmE >= 123) && (cmE < 128)) p = p125;
if((cmE >= 128) && (cmE < 133)) p = p130;
if((cmE >= 133) && (cmE < 138)) p = p135;
if((cmE >= 138) && (cmE < 143)) p = p140;
if((cmE >= 143) && (cmE < 148)) p = p145;
if((cmE >= 148) && (cmE < 153)) p = p150;
if((cmE >= 153) && (cmE < 158)) p = p155;
if((cmE >= 158) && (cmE < 163)) p = p160;
if((cmE >= 163) && (cmE < 168)) p = p165;
if((cmE >= 168) && (cmE < 173)) p = p170;
if((cmE >= 173) && (cmE < 178)) p = p175;
if((cmE >= 178) && (cmE < 183)) p = p180;
if((cmE >= 183) && (cmE < 188)) p = p185;
if((cmE >= 188) && (cmE < 193)) p = p190;
if((cmE >= 193) && (cmE < 198)) p = p195;
if((cmE >= 198) && (cmE < 203)) p = p200;

p->GetParticlesOfEvent(partVec, cmV);

Projectile->nucl[ip].ncoll++;
        Target->nucl[it].ncoll++;
      }

      // check if there was a collisions with very small cross section
      // (only black disk cross section here)
      if (dxy2 < sigma_nn_hard_fm2/pi) {

       if (NcollHard < NcollHardMax) {
        jpc[NcollHard].ip = ip;
        jpc[NcollHard].it = it;
        jpc[NcollHard].x = (Projectile->nucl[ip].x + Target->nucl[it].x)/2;
        jpc[NcollHard].y = (Projectile->nucl[ip].y + Target->nucl[it].y)/2;

        NcollHard++;
       }
       else {
       cout << "WARNING:
```

```
       counter for coll. that create a J/Psi out of bounds:"
            << "NCollHard = " << NcollHard << endl;
       cout << "J/Psi survival probability will be useless !!!" << endl;
}
      }

    }
  }

  // get number of participants
  for (int ip=0; ip<Projectile->A; ip++) {
    if (Projectile->nucl[ip].ncoll != 0) {

      // for calculation of pathlength L
      NSumSubsColl += int(Projectile->nucl[ip].ncoll
      * (Projectile->nucl[ip].ncoll - 1)/2.);

      Projectile->part[NpartProj] = &(Projectile->nucl[ip]);
      Part[Npart] = &(Projectile->nucl[ip]);
      Npart++;
      NpartProj++;
    }
  }

  for (int it=0; it<Target->A; it++) {
    if (Target->nucl[it].ncoll != 0) {

      // for calculation of pathlength L
      NSumSubsColl += int(Target->nucl[it].ncoll
      * (Target->nucl[it].ncoll - 1)/2.);

      Target->part[NpartTarg] = &(Target->nucl[it]);
      Part[Npart] = &(Target->nucl[it]);
      Npart++;
      NpartTarg++;
    }
  }

  //
  // determine how many of the created J/Psi survived
  //

  // negative absorption cross means: skip this part to save CPU time
  if (sigma_jpsi_abs_fm2 >= 0) {
```

```
    for (int ijpc=0; ijpc < NcollHard; ijpc++) {

      bool bAbsorbed = false;

      // check Projectile
      for (int ip=0; ip<Projectile->A; ip++) {

if (jpc[ijpc].ip != ip && Projectile->nucl[ip].z <
    Projectile->nucl[jpc[ijpc].ip].z) {

  // squared transverse distance of nucleon and J/Psi
  dxy2 = pow((Projectile->nucl[ip].x - jpc[ijpc].x),2) +
    pow((Projectile->nucl[ip].y - jpc[ijpc].y),2);

  if (dxy2 < sigma_jpsi_abs_fm2/pi) {
    bAbsorbed = true;
    break;
  }

}
      }

      if (!bAbsorbed) {

// check Target
for (int it=0; it<Target->A; it++) {

  if (jpc[ijpc].it != it && Target->nucl[it].z >
      Target->nucl[jpc[ijpc].it].z) {

    // squared transverse distance of nucleon and J/Psi
    dxy2 = pow((Target->nucl[it].x - jpc[ijpc].x),2) +
      pow((Target->nucl[it].y - jpc[ijpc].y),2);

    if (dxy2 < sigma_jpsi_abs_fm2/pi) {
      bAbsorbed = true;
      break;
    }

  }
}
      }
```

```
      if (!bAbsorbed) NcollHardSurv++;

    }

  }

  if (Npart > 0) CalcEccentricityAndArea();

  //4/17/07: partVec iteration moved here so that npart, impact parameter,
  //         and eccentricity can be included in tree

  int nstore = 0;
  vector<particleStore>::iterator pPartVec = partVec.begin();

  // Set the number of particles created for this event
  hdrAA->SetNparticles(partVec.size());

  while ( pPartVec < partVec.end() ) {

    particle->SetKF(pPartVec->getPid());
    particle->SetEnergy(pPartVec->getE());
    particle->SetPx(pPartVec->getPx());
    particle->SetPy(pPartVec->getPy());
    particle->SetPz(pPartVec->getPz());

    new ((*part)[nstore++]) TMCParticle(*particle);
    pPartVec++;
  }

  partVec.clear();

  return (Ncoll == 0) ? 0 : 1;
} //end of collide(i)


//! This function calculates the eccentricity and the overlap area
//! of the participating nucleons in the transverse plane.
void AACollision::CalcEccentricityAndArea() {

  // ------------------------------------------------------------
  // standard eccentricity calculated in fixed coordinate system
  // (impact parameter vector defines x-axis)
  // ------------------------------------------------------------
```

```
// calculate origin for standard eccentricity calculation
double x0 = b / 2.;
double y0 = 0.;

int    iCount = 0;
double sumx   = 0.;
double sumy   = 0.;
double sumx2  = 0.;
double sumy2  = 0.;

for (int iPart=0; iPart<Npart; iPart++) {
  sumx  += Part[iPart]->x;
  sumy  += Part[iPart]->y;
  sumx2 += pow(Part[iPart]->x - x0, 2);
  sumy2 += pow(Part[iPart]->y - y0, 2);
  iCount++;
}

double EccNum = sumy2 - sumx2;
double EccDen = sumy2 + sumx2;
if (EccDen != 0) Eccentricity = EccNum/EccDen;
else             Eccentricity = -9999.;

// ---------------------------------------------------------
// overlap area
// ---------------------------------------------------------
double meanx2 = sumx2 / double(iCount);
double meany2 = sumy2 / double(iCount);
OverlapArea = pi * sqrt(meanx2 * meany2);

// ---------------------------------------------------------
// eccentricity without reference to external coordinate system
// ---------------------------------------------------------

// step 1: calculate major axis of participant distribution in
//         the transverse plane

// center of gravity of participant distribution
double xmean = sumx / iCount;
double ymean = sumy / iCount;

// reset counting variables to zero
sumx2 = 0.;
sumy2 = 0.;
```

```
double sumxy = 0.;

for (int iPart=0; iPart<Npart; iPart++) {
  double xs = Part[iPart]->x - xmean;
  double ys = Part[iPart]->y - ymean;
  sumx2 += xs*xs;
  sumy2 += ys*ys;
  sumxy += xs*ys;
}

// calculate rotation angle phi of the coordinate system
// which maximizes the eccentricity
double alpha  = atan2(2.*sumxy,sumx2-sumy2);
if (alpha < 0.) alpha += 2*pi;
double phi = alpha/2. - pi/2.;
double cosphi = cos(phi);
double sinphi = sin(phi);

// step 2: calculate eccentricity in rotated system
double EccNumWERCS =   sinphi*sinphi*(sumx2-sumy2)
                     + cosphi*cosphi*(sumy2-sumx2)
                     - 4.*sinphi*cosphi*sumxy;
double EccDenWERCS = sumx2 + sumy2;

if (EccDenWERCS != 0) EccentricityWERCS = EccNumWERCS/EccDenWERCS;
else EccentricityWERCS = -9999.;

// ------------------------------------------------------------
// v2 of participant distribution
// ------------------------------------------------------------
double sumv2 = 0.;

// loop of participant that is skipped in the calculation
// of the 'reaction plane' in order to avoid auto-correlations
for (int iPartSkip=0; iPartSkip<Npart; iPartSkip++) {

  // center-of-gravity without participant iPartSkip
  double xmeantmp = (sumx - Part[iPartSkip]->x)/(Npart-1);
  double ymeantmp = (sumy - Part[iPartSkip]->y)/(Npart-1);

  // now determine 'reaction plane' of the participants
  // without participant iPartSkip
  double sumx2tmp = 0.;
  double sumy2tmp = 0.;
```

```
  double sumxytmp = 0.;

  for (int iPart=0; iPart<Npart; iPart++) {
    if (iPart != iPartSkip) {
      double xs = Part[iPart]->x - xmeantmp;
      double ys = Part[iPart]->y - ymeantmp;
      sumx2tmp += xs*xs;
      sumy2tmp += ys*ys;
      sumxytmp += xs*ys;
    }
  }

  // calculate angle of participant iPartSkip with respect
  // to 'reaction plane' of the other participants
  if (sumx2tmp-sumy2tmp != 0.) {

    // this formula corresponds to the determination of
    // the 'reaction plane' with weight equal to the distance
    // to the center-of-gravity squared
    double alpha  = atan2(2.*sumxytmp,sumx2tmp-sumy2tmp);
    if (alpha < 0.) alpha += 2*pi;

    // calculate angle of the participant that was left out
    double xs = Part[iPartSkip]->x - xmeantmp;
    double ys = Part[iPartSkip]->y - ymeantmp;

    double phiSkip = atan2(ys,xs);
    if (phiSkip < 0.) phiSkip += 2*pi;

    // calculate 2*phiskip - 2*phi = 2*phiskip - alpha
    double phiDiff = 2*phiSkip - alpha;

    sumv2 += cos(phiDiff);
  }
}

// set v2 variable
v2Part = sumv2 / Npart;

// check: participant v2 with auto-correlation
sumv2 = 0.;
double sumeps = 0.;

for (int iPart=0; iPart<Npart; iPart++) {
```

```
    double xs = Part[iPart]->x - xmean;
    double ys = Part[iPart]->y - ymean;

    double phiPart = atan2(ys,xs);
    if (phiPart < 0.) phiPart += 2*pi;

    // calculate 2*phiskip - 2*phi = 2*phiskip - alpha
    double phiDiff = 2*phiPart - alpha;

    sumv2 += cos(phiDiff);

    // futher check: calculate v2PartAuto in a different way
    sumeps += (xs*xs - ys*ys)/(xs*xs + ys*ys);

  }

  double v2PartAuto = sumv2 / Npart;
  double v2PartAutoCheck = sumeps / Npart;
  // cout << "v2=" << v2Part << ", v2PartAuto=" << v2PartAuto
  //      << ", v2PartAutoCheck=" << v2PartAutoCheck << endl;


};

//! Returns the eccentricity
double AACollision::GetEccentricity() {
  return Eccentricity;
};

//! Returns the eccentricity calculated without external reference
//! coordinate system
double AACollision::GetEccentricityWERCS() {
  return EccentricityWERCS;
};

//! Returns overlap area in the transverse plane
double AACollision::GetOverlapArea() {
  return OverlapArea;
};

//! Returns v2 of the participant distribution
double AACollision::GetPartV2() {
  return v2Part;
```

```
};

//! Initialize parameters for Gaussian NN overlap function
void AACollision::InitGausOvlPara() {

  // Define parameters for Gaussian NN overlap function.
  // We leave the width of the overlapfunction constant and
  // change the absolute normalization according to the
  // selected inelastic NN cross section. This recipe fails
  // for large cross sections (LHC?). In this case, one would
  // have to increase the width of the overlap function (fGausOvlP2)

  // width of Gaussian overlap function
  fGausOvlP2  = 0.89; // fm^2

  // calculate first parameter so that integral corresponds
  // to total inelastic cross section
  double fTmp = 4. - sigma_nn_inel_fm2 * fGausOvlP2 * 2. / pi;
  if (fTmp > 0. && fTmp < 4.) {
    fGausOvlP1 = 2. - sqrt(fTmp);
  }
  else {
   cout << "AACollision::InitGausOvlPara: Couldn't initialize parameters."
   << endl;
   exit(-1);
  }
  // cout << "fGausOvlP1 = " << fGausOvlP1 << ", fGausOvlP2 = "
  << fGausOvlP2 << endl;
};

//! Sets the impact parameter for the AA collision.
void AACollision::SetImpactParameter(const double& ImpactPara) {
  b = ImpactPara;
  Projectile->SetCoordinates(b,0.,-30.);
  Target->SetCoordinates(0.,0.,0.);
}

//! Returns the AA impact parameter
double AACollision::GetImpactParameter() {
  return b;
}

//! Returns Npart and Ncoll
void AACollision::GetNpartNcoll(int& npproj, int& nptarg,
```

```
                                int& nc, int& nchard, int& nchardsurv,
int& nsubs) {
  npproj     = NpartProj;
  nptarg     = NpartTarg;
  nc         = Ncoll;
  nchard     = NcollHard;
  nchardsurv = NcollHardSurv;
  nsubs      = NSumSubsColl;
}


//! returns impact parameter of the i-th target participant
double AACollision::GetImpactParTargPart(int iPart) {

  if (iPart >= 0 && iPart < NpartTarg) {

    // coordinates of the nucleon
    double xNucleon = Target->part[iPart]->x;
    double yNucleon = Target->part[iPart]->y;

    // coordinates of the projectile nucleus
    double xNucleus = Projectile->x;
    double yNucleus = Projectile->y;

    double bNucleon = sqrt((yNucleon - yNucleus) * (yNucleon - yNucleus) +
                           (xNucleon - xNucleus) * (xNucleon - xNucleus));

    return bNucleon;
  }
  else {
    // cout << "Warning:: GetImpactParTargPart: out of range, iPart = "
    //      << iPart << endl;
    return -1;
  }
}

//! returns impact parameter of the i-th projectile participant
double AACollision::GetImpactParProjPart(int iPart) {

  if (iPart >= 0 && iPart < NpartProj) {

    // coordinates of the nucleon
    double xNucleon = Projectile->part[iPart]->x;
    double yNucleon = Projectile->part[iPart]->y;
```

```
    // coordinates of the target nucleus
    double xNucleus = Target->x;
    double yNucleus = Target->y;

    double bNucleon = sqrt((yNucleon - yNucleus) * (yNucleon - yNucleus)+
                           (xNucleon - xNucleus) * (xNucleon - xNucleus));

    return bNucleon;
  }
  else {
    // cout << "Warning:: GetImpactParProjPart: out of range, iPart = "
    //      << iPart << endl;
    return -1;
  }
}

//! Returns the simulated PHENIX ZDC energy
void AACollision::GetPhenixZDCEnergy(double& eZDC_n, double& eZDC_s,
  double& eZDCpp_n, double& eZDCpp_s, int& ZDCTrig) {

  // find all spectator neutrons and reject a certain fraction
  // according to probabiliy derived from NA49 paper:
  // neutron loss prob. as function of b from NA49 paper

  double b = GetImpactParameter();
  double CoalescenesLossProb;

  // the coalescenes loss parameterrization below is valid
  // for Pb (or Au). In order to calculate an effective impact
  // parameter b for smaller nuclei (e.g. Cu) we assume a
  // simple scaling (which is not well justified)
  double bmax_Au = 2 * (6.38 + 3 * 0.54);
  double bmax    = Projectile->ws_radius + 3. * Projectile->ws_diff +
                   Target->ws_radius + 3. * Target->ws_diff;

  // scale impact parameter for nuclei lighter than Au for
  // sampling of coaslescenes loss function
  double beff = b * (bmax_Au / bmax);

  if(ZdcCheck) {
    // param. I (old)
    CoalescenesLossProb = 0.1443 + beff*0.0422;
  }
```

```
else {
  // param. II (default)
  double p0 = 0.3305;
  double p1 = 0.0127;
  double p2 = 17.;
  double p3 = 2.;
  CoalescenesLossProb = p0+b*p1+exp((beff-p2)/p3);
}

// neutron loss due to limited ZDC acceptance
double AccLossProb = 1 - 1./1.4;

// total neutron loss probability
double lossProbability = 1 - (1 - CoalescenesLossProb)*(1 - AccLossProb);

// get zero degree Neutrons from Projectile
int nZDNeutronsProj = 0;
for (int ip=0; ip<Projectile->A; ip++) {
  if (Projectile->nucl[ip].type  == neutron &&
      Projectile->nucl[ip].ncoll == 0       &&
      gRandom->Rndm() > lossProbability        ) nZDNeutronsProj++;
}

// get zero degree Neutrons from Target
int nZDNeutronsTarg = 0;
for (int it=0; it<Target->A; it++) {
  if (Target->nucl[it].type  == neutron &&
      Target->nucl[it].ncoll == 0       &&
      gRandom->Rndm() > lossProbability    ) nZDNeutronsTarg++;
}

// determine ZDC trigger status
ZDCTrig = (nZDNeutronsProj>0 && nZDNeutronsTarg>0) ? 1 : 0;

// calculate energy of neutrons in forward direction
double eZDCProj = ecms/2.* nZDNeutronsProj;
double eZDCTarg = ecms/2.* nZDNeutronsTarg;


// smear energy signal according to ZDC resolution:
// sigma_E / E = 218 % / sqrt(E)
double eZDCProjSmeared = gRandom->Gaus(eZDCProj, 2.18*sqrt(eZDCProj));
double eZDCTargSmeared = gRandom->Gaus(eZDCTarg, 2.18*sqrt(eZDCTarg));
```

```
  // return values without contribution from produced particles
  eZDC_n = eZDCProjSmeared;
  eZDC_s = eZDCTargSmeared;

  // get total ZDC energy without participant contribution
  // (first return value)
  // eZDC = eZDCProjSmeared + eZDCTargSmeared;

  // Now the same with an estimation of the contribution
  // of participants (and produced particles) to ZDC energy.
  // estimate probability that participant ends up in ZDC acceptance
  double PartContribProb = 0.02;

  // estimate mean energy carried by a participating nucleon
  // (should in principle be less than beam energy)
  double MeanEPart = ecms/2.;

  // get ZDC energy coming from participants
  double eZDCPartProj = 0;
  for (int ipp=0; ipp<NpartProj; ipp++) {
    if(gRandom->Rndm() < PartContribProb) eZDCPartProj += MeanEPart;
  }
  double eZDCPartTarg = 0;
  for (int ipt=0; ipt<NpartTarg; ipt++) {
    if(gRandom->Rndm() < PartContribProb) eZDCPartTarg += MeanEPart;
  }

  // add participant contribution
  eZDCProj += eZDCPartProj;
  eZDCTarg += eZDCPartTarg;

  // smear energy signal according to ZDC resolution:
  // sigma_E / E = 218 % / sqrt(E)
  double eZDCPlusProjPart = gRandom->Gaus(eZDCProj, 2.18*sqrt(eZDCProj));
  double eZDCPlusTargPart = gRandom->Gaus(eZDCTarg, 2.18*sqrt(eZDCTarg));

  // ZDC energy distribution plus estimated participant contribution
  // eZDCpp = eZDCPlusProjPart + eZDCPlusTargPart;

  // return value
  eZDCpp_n = eZDCPlusProjPart;
  eZDCpp_s = eZDCPlusTargPart;

}
```

```
//! Returns the value of the simulated centrality variable
//! (e.g. Q_BBC or N_PC1).
//! This quantity is obtained by sampling the NBD several times, as
//! determined by Npart^alpha
int AACollision::GetNhitsFromNBD() {

  // Allow non-linear scaling of BBC hit multiplicity with N_part:
  // Map Npart on N_cluster ~ N_part^alpha and sample NBD distribution
  // N_cluster times. Linear scaling with N_part ist then special case
  // alpha = 1

  int nHits = 0;
  int nClus = int( pow(NpartProj+NpartTarg, alpha) + 0.5);

  for (int i=0; i < nClus; i++) {
    nHits += nbd.getRandom();
  }

  return nHits;

}


int main(int argc, char** argv) {

  if (argc != 2) {
    cout << "Usage: glauber_mc <inputfile>" << endl;
    exit(-1);
  }


  // Glauber parameters
  double A_Nucleus1    = 0.;
  double Z_Nucleus1    = 0.;
  double A_Nucleus2    = 0.;
  double Z_Nucleus2    = 0.;
  double wsradius1     = 0.;
  double wsdiff1       = 0.;
  double wsradius2     = 0.;
  double wsdiff2       = 0.;
  double bmin          = 0.;
  double bmax          = 0.;
  double cmsEnergy     = 0.;
```

```
double sginel        = 0.;
double sgtot         = 0.;
double sg_jpsi_abs   = 0.;
double nnOverlapFunc = 0.;
double NucleonsWithHardCore = 0.;
double ZdcCheck      = 0.;
double nbd_mu        = 0.;
double nbd_k         = 0.;
double alpha         = 0.;
double fNevents      = 0.;
double eLossfraction = 0.; //(RZ, added 6/21/07)
double fNprint       = 0.;

string sfout;

//
// read Glauber settings
//
ifstream fgs(argv[1]);

// Nuclear Mass and Z of Nucleus 1
GetNumValue(fgs,A_Nucleus1);
GetNumValue(fgs,Z_Nucleus1);

// Nuclear Mass and Z of Nucleus 2
GetNumValue(fgs,A_Nucleus2);
GetNumValue(fgs,Z_Nucleus2);

// Woods-Saxon Parameters for nucleus 1
GetNumValue(fgs,wsradius1);
GetNumValue(fgs,wsdiff1);

// Woods-Saxon Parameters for nucleus 2
GetNumValue(fgs,wsradius2);
GetNumValue(fgs,wsdiff2);

// impact parameter range
GetNumValue(fgs,bmin);
GetNumValue(fgs,bmax);

// cms energy
GetNumValue(fgs, cmsEnergy);

// inel. and total cross section
```

```
GetNumValue(fgs,sginel);
GetNumValue(fgs,sgtot);

// J/Psi absorption cross section in mb
GetNumValue(fgs,sg_jpsi_abs);

// get the nn overlap function type
// 0 = black disk
// 1 = gray disk
// 2 = gaussian
GetNumValue(fgs,nnOverlapFunc);

// Nucleons with hard core (yes/no = 1/0)
GetNumValue(fgs,NucleonsWithHardCore);

// ZDC check on/off = 1/0
GetNumValue(fgs,ZdcCheck);

// parameters for negative binomial distribution used in
// BBC simulation
GetNumValue(fgs,nbd_mu);
GetNumValue(fgs,nbd_k);

// power alpha that determines scaling of BBC hit multiplicity
// with n_part
GetNumValue(fgs,alpha);

// Number of events, E. Loss Fraction (RZ, 6/21/07), and print frequency
GetNumValue(fgs,fNevents);
GetNumValue(fgs,eLossfraction);
GetNumValue(fgs,fNprint);

// Name of root output file
GetStringValue(fgs,sfout);

fgs.close();

//
// set Glauber parameters and print values
//

cout << ">>> ---------------------------" << endl;
cout << ">>> PHENIX Glauber calculation:" << endl;
cout << ">>> ---------------------------" << endl;
```

```
//out << ">>> ---------------------------" << endl;
//out << ">>> PHENIX Glauber calculation:" << endl;
//out << ">>> ---------------------------" << endl;

// define target and projectile nucleus
nucleus* A;
if (wsradius1 < 0 || wsdiff1 < 0) {
  A = new nucleus((int)A_Nucleus1,(int)Z_Nucleus1);
}
else {
  A = new nucleus((int)A_Nucleus1,(int)Z_Nucleus1,wsradius1,wsdiff1);
}

nucleus* B;
if (wsradius2 < 0 || wsdiff2 < 0) {
  B = new nucleus((int)A_Nucleus2,(int)Z_Nucleus2);
}
else {
  B = new nucleus((int)A_Nucleus2,(int)Z_Nucleus2,wsradius2,wsdiff2);
}

cout << ">>> A_Nucleus1:                      " << A_Nucleus1 << endl;

cout << ">>> Z_Nucleus1:                      " << Z_Nucleus1 << endl;

cout << ">>> A_Nucleus2:                      " << A_Nucleus2 << endl;

cout << ">>> Z_Nucleus2:                      " << Z_Nucleus2 << endl;

cout << ">>> Woods-Saxon radius nucleus1(fm): " << wsradius1 << endl;

cout << ">>> Woods-Saxon diff. nucleus1 (fm): " << wsdiff1 << endl;

cout << ">>> Woods-Saxon radius nucleus2(fm): " << wsradius2 << endl;

cout << ">>> Woods-Saxon diff. nucleus2 (fm): " << wsdiff2 << endl;

cout << ">>> bmin (fm):                       " << bmin << endl;

cout << ">>> bmax (fm):                       " << bmax << endl;

// function for impact parameter sampling
if (bmin > bmax) {
```

```
  cout << "error: bmin > bmax" << endl;
  exit(-1);
}

TF1* fImpact = new TF1("fImpact","x",bmin,bmax);

// define new collision
AACollision* c = new AACollision(A,B);

// cms energy
c->SetCMSEnergy(cmsEnergy);
cout << ">>> cms-energy (GeV):                " << cmsEnergy << endl;

// cross sections
c->SetInelNNCrossSection(sginel);
c->SetTotalNNCrossSection(sgtot);
cout << ">>> inel. nn x-section (mb):         " << sginel << endl;
cout << ">>> total nn x-section (mb):         " << sgtot << endl;

// J/Psi absorption cross section
c->SetJPsiAbsCrossSection(sg_jpsi_abs);
cout << ">>> J/Psi abs. x-section (mb):       " << sg_jpsi_abs << endl;

// the switches that specify details of the simulation
c->SetNNOverlapFunc((int)nnOverlapFunc);
switch (int(nnOverlapFunc)) {
case 0:
    cout << ">>> nn overlap function:             black disk" << endl;
    break;
case 1:
    cout << ">>> nn overlap function:             gray disk" << endl;
    break;
case 2:
    cout << ">>> nn overlap function:             gaussian" << endl;
    c->InitGausOvlPara(); // only works if NN cross section is defined
    break;
}

if (NucleonsWithHardCore != 0) {
  cout << ">>> Nucleons with hard core:         yes" << endl;
  c->SetHardCoreNucleons(true);
}
else {
  cout << ">>> Nucleons with hard core:         no" << endl;
```

```
  c->SetHardCoreNucleons(false);
}

if (ZdcCheck != 0) {
  cout << ">>> ZdcCheck:                          yes" << endl;
  c->SetZdcCheck(true);
}
else {
  cout << ">>> ZdcCheck:                          no" << endl;
  c->SetZdcCheck(false);
}

// parameters for negative binomial distribution used in
// BBC simulation
cout << ">>> NBD parameter mu:                     " << nbd_mu << endl;
cout << ">>> NBD parameter k:                      " << nbd_k << endl;
c->SetPhenixBBCNbdParams(nbd_mu, nbd_k);

// power alpha that determines the scaling of the BBC hit
// multiplicity with N_part
cout << ">>> Npart scaling power alpha:            " << alpha << endl;
c->SetPhenixBBCMultScalingPower(alpha);

// Energy Loss Fraction for NN Collisions (RZ, added 6/21/07):
cout << ">>> Energy Loss Fraction:             " << eLossfraction << endl;
c->SetELossFraction(eLossfraction);

cout << ">>> output file:                      " << sfout << endl;

cout << endl;

// set cross-section for hard processes
// (not really needed, just a check)
c->SetHardNNCrossSection(40./100.);


// --------------   5/4/07, Robert Zaballa:  ----------------------
//Histograms for AA events themselves without particle production
//parameters.

// open new root file
TFile* myfile = new TFile(sfout.c_str(),"RECREATE");

/*
```

```
// book some histograms
TH1F* hImpactAll = new TH1F("hImpactAll",
                            "Impact parameter (all collisions)",
                            200,-0.005,19.995);
TH1F* hImpactIn  = new TH1F("hImpactIn",
                            "Impact parameter (Ncoll >= 1)",
                            200,-0.005,19.995);
TH1F* hNpartvsb  = new TH1F("hNpartvsb","N?part! vs. b",
                            200,-.005,19.995);
TH1F* hNpart     = new TH1F("hNpart","dN/dN?part! (Ncoll >= 1)",
                            200,-0.5,999.5);
TH1F* hNpZDCTrig = new TH1F("hNpZDCTrig",
                            "dN/dN?part! (Ncoll >= 1) && ZDC Trigger",
                            500,-0.5,499.5);
TH1F* hNpBBCTrig = new TH1F("hNpBBCTrig",
                            "dN/dN?part! (Ncoll >= 1) && BBC Trigger",
                            500,-0.5,499.5);
TH1F* hpTrigBBC  = new TH1F("hpTrigBBC","BBC trigger prob.",
                            100,0.,1.);
TH1F* hNcollvsb  = new TH1F("hNcollvsb","N?coll! vs. b",
                            200,-.005,19.995);
TH1F* hNcoll     = new TH1F("hNcoll","dN/dN?coll! (Ncoll >= 1)",
                            200,-0.5,999.5);

//------------Additional Histograms added on 1/19/07-----------
TH1F* hEccent  = new TH1F("hEccent","Eccentricity",
                          100,0.0,1.0);
TH1F* hEccentWERCS  = new TH1F("hEccentWERCS","EccentricityWERCS",
                        100,0.0,1.0);
TH1F* hArea     = new TH1F("hArea","Overlap Area",
                           100,-0.5,49.5);
TH1F* hv2     = new TH1F("hv2","v2",
                         100,0.0,1.0);
//------------------------------------------------------------
*/

int icount = 0;

// define number of reactions to simulate
const int nAAColl = int(fNevents);
const int nPrint  = int(fNprint);

Int_t    npart;
Int_t    ZDCTrig, BBCTrig;
```

```
Double_t pTrigBBC;
Double_t eZDC, eZDC_n, eZDC_s, eZDCNeut;
Double_t eZDCNeutPP, qBBC;

// variables which are written into tree
Double_t b;          // impact parameter
Int_t    npartproj;  // number of projectile participants
Int_t    nparttarg;  // number of target participants
Int_t    ncoll;      // number of inelastic nucleon-nucleon collisions
Int_t    ncollhard;  // number of hard (sigma_hard << sigma_inel)
                     // NN collisions
Int_t    ncollhardsurv;     // ...
Int_t    nsubs;      // needed for calculation of J/Psi path length
Double_t eZDCNeut_n; // simulated ZDC north signal (A+A)
Double_t eZDCNeut_s; // simulated ZDC south signal (A+A)
Double_t eZDCNeutPP_n; // ZDC north signal + estimated participant
                       // contribution
Double_t eZDCNeutPP_s; // ZDC south signal + estimated participant
                       // contribution
Int_t    nHitsNBD;   // number of hits of centrality variable
                     // (e.g. BBC or PC1)
// Double_t qBBC_n;    // total charge signal in pC in BBC north
// Double_t qBBC_s;    // total charge signal in pC in BBC south
Double_t Eccentricity; // eccentricity of the 2d-participant distribution
Double_t EccentricityWERCS; // eccentricity without external reference
Double_t v2Part;     // v2 of the participant distribution
Double_t OverlapArea;// overlap area
Double_t bprojpart1; // impact parameter of first projectile participant
                     // (for d+Au)
Double_t bprojpart2; // impact parameter of second projectile participant
                     // (for d+Au)

/*
//define output tree  TTree* gl = new TTree("gl","Glauber tree");
TTree* gl = new TTree("gl","Glauber tree");

gl->Branch("b",  &b,  "b/D");
gl->Branch("npartproj", &npartproj,  "npartproj/I");
gl->Branch("nparttarg", &nparttarg,  "nparttarg/I");
gl->Branch("ncoll", &ncoll,  "ncoll/I");
gl->Branch("ncollhard", &ncollhard,  "ncollhard/I");
gl->Branch("ncollhardsurv", &ncollhardsurv,  "ncollhardsurv/I");
gl->Branch("nsubs", &nsubs,  "nsubs/I");
gl->Branch("ezdc_n", &eZDCNeut_n,  "ezdc_n/D");
```

```
gl->Branch("ezdc_s", &eZDCNeut_s,  "ezdc_s/D");
gl->Branch("ezdcpp_n", &eZDCNeutPP_n,  "ezdcpp_n/D");
gl->Branch("ezdcpp_s", &eZDCNeutPP_s,  "ezdcpp_s/D");
gl->Branch("nHitsNBD", &nHitsNBD,  "nhitsnbd/I");
// gl->Branch("qbbc_n", &qBBC_n,  "qbbc_n/D");
// gl->Branch("qbbc_s", &qBBC_s,  "qbbc_s/D");
gl->Branch("eccentr", &Eccentricity, "eccentr/D");
gl->Branch("eccentrwer", &EccentricityWERCS, "eccentrwer/D");
gl->Branch("v2part", &v2Part, "v2part/D");
gl->Branch("area", &OverlapArea, "area/D");
gl->Branch("bprojpart1", &bprojpart1, "bprojpart1/D");
gl->Branch("bprojpart2", &bprojpart2, "bprojpart2/D");
*/

//Branches for GLtree (for Produced Particles)

GLtree->Branch("event", "headerAA", &hdrAA, 2048, 1);
GLtree->Branch("part", &part, 2048, 1);

gRandom->SetSeed(0);

//
// A+A collision loop
//
for (int i=0; i<nAAColl; i++) {

  //
  // simulate collision
  //

  // get impact parameter b, allow bmim == bmax == b
  if (bmin != bmax) {
    b = fImpact->GetRandom();
  }
  else {
    b = bmin;
  }

  c->SetImpactParameter(b);
  c->Collide(i);
  c->GetNpartNcoll(npartproj, nparttarg, ncoll,
   ncollhard, ncollhardsurv, nsubs);
  npart = npartproj + nparttarg;
  Eccentricity = c->GetEccentricity();
```

```
EccentricityWERCS = c->GetEccentricityWERCS();
v2Part = c->GetPartV2();
OverlapArea  = c->GetOverlapArea();

// Fill up the event branch!
hdrAA->SetEvt(i);
hdrAA->SetCME(cmsEnergy);
hdrAA->SetELossfraction(eLossfraction);
hdrAA->SetNpartTarg(nparttarg);
hdrAA->SetNpartProj(npartproj);
hdrAA->SetNNcoll(ncoll);
hdrAA->SetImpactParam(b);
hdrAA->SetEccent(Eccentricity);
hdrAA->SetV2(v2Part);
hdrAA->SetOverlapArea(OverlapArea);

// Fill GLtree after completing event and particle branches
GLtree->Fill();

// reset the part vector
part->Clear();

/*
// Commented out for saving memory usage

// fill histograms
hImpactAll->Fill(b);

if (ncoll > 0) {
hImpactIn->Fill(b);
hNpartvsb->Fill(b,double(npart));
hNpart->Fill(double(npart));
hNcollvsb->Fill(b,double(ncoll));
hNcoll->Fill(double(ncoll));

//----fill Additional Histograms added on 1/19/07----
hEccent->Fill(Eccentricity);
hEccentWERCS->Fill(EccentricityWERCS);
hArea->Fill(OverlapArea);
hv2->Fill(v2Part);
*/

//-------------------------------------------------
```

```
  // get ZDC energy w and w/o participant contribution
  c->GetPhenixZDCEnergy(eZDCNeut_n,eZDCNeut_s,
eZDCNeutPP_n,eZDCNeutPP_s,ZDCTrig);
  eZDCNeut   = eZDCNeut_n + eZDCNeut_s;
  eZDCNeutPP = eZDCNeutPP_n + eZDCNeutPP_s;

  /*

  // check if ZDC would have triggered
  if (ZDCTrig) hNpZDCTrig->Fill(double(npart));

  */
  // number of hits for centrality variable from sampling of NBD
  nHitsNBD = c->GetNhitsFromNBD();

  // PHENIX BBC charge
  // c->GetPhenixBBCInfo(nHitBbc_n,nHitBbc_s,qBBC_n,qBBC_s,
  //                     pTrigBBC,BBCTrig);
  // qBBC = qBBC_n + qBBC_s;

  // check if BBC would have triggered
  // if (BBCTrig) hNpBBCTrig->Fill(double(npart));

  // fill histogram with BBC trigger probability
  // hpTrigBBC->Fill(pTrigBBC);

  // fill impact parameters of the first two projectile participants
  // (--> needed in case the projectile is a deuteron)
  // function return -1 if participant does not exist
  bprojpart1 = c->GetImpactParProjPart(0);
  bprojpart2 = c->GetImpactParProjPart(1);

  /*
  // fill the Glauber tree
  icount++;
  gl->Fill();
  } //end of if(ncoll > 0)
  */


  // print information
  if (i%nPrint == 0) {
```

```
      cout << "AA Event: " << i;
      cout << ", b = " << b;
      cout << ", ZDC energy: " << eZDCNeut << " GeV";
      cout << ", Npart = " << npart;
      cout << ", Ncoll = " << ncoll << endl;
    }
    eZDCNeut = 0;
  } //end of AA collision loop

  GLtree->Write();

  // free allocted memory
  delete c;
  delete fImpact;

  // save the histograms
  myfile->Write();
  myfile->Close();

} //end of int main()


void GetNumValue(ifstream& f, double& value) {

  //
  // read numerical value from Glauber input file. Skip
  // comments and whitespace.
  //

  bool eof = false;
  bool valueRead = false;
  char ch;

  do {

    // skip whitespace
    do {
      if(!f.get(ch)) eof=true;
    } while (eof==false && isspace(ch));

    // skip comments
    if (ch=='!') {
      do {
        if(!f.get(ch)) eof=true;
```

```
      } while (ch!='\n');
    }
    else {
      f.putback(ch);
      f >> value;
      valueRead = true;
    }

  } while(eof==false && valueRead == false);

}


void GetStringValue(ifstream& f, string& value) {

  //
  // read string value from Glauber input file. Skip
  // comments and whitespace.
  //

  bool eof = false;
  bool valueRead = false;
  char ch;

  do {

    // skip whitespace
    do {
      if(!f.get(ch)) eof=true;
    } while (eof==false && isspace(ch));

    // skip comments
    if (ch=='!') {
      do {
        if(!f.get(ch)) eof=true;
      } while (ch!='\n');
    }
    else {
      f.putback(ch);
      f >> value;
      valueRead = true;
    }

  } while(eof==false && valueRead == false);
```

```
}
```

From the file PythiaEventAccessor.h:

```
/*
 * This is Robert's Pythia Event Accessor class.
 *
 * This is a second version where the class only
 * needs to read the input energy from the Glauber
 * model N-N collision. The appropriate root file
 * will then be accessed based on rounding the input
 * energy to the nearest value (integer value) that
 * corresponds to a root file. The events in the
 * root file are then accessed.
 *
 * 2/12/2007 Created
 */
#include <iostream>
#include <TMCParticle6.h>
#include <TFile.h>
#include <TTree.h>
#include <TLeaf.h>
#include <TH1I.h>
#include <TH1F.h>
#include <string>
#include <sstream>
#include "header.h"
#include "particleStore.h"

using namespace std;

class PythiaEventAccessor
{
private:

  int cmsEnergy;
  int nevents;
  int Energy;                // rounded cms energy value
  int CurrentEvent;          //the event called

  string filename;           //Name of input root file

  TFile *f;
```

```
  TTree *tree;
  TClonesArray *particles;

  TBranch *partBranch;

  TBranch *id;
  TBranch *energy;  // energy of this produced particle
  TBranch *px;
  TBranch *py;
  TBranch *pz;

  TLeaf *leafid;
  TLeaf *leafenergy;
  TLeaf *leafpx;
  TLeaf *leafpy;
  TLeaf *leafpz;

public:

  PythiaEventAccessor(double e);
  PythiaEventAccessor();
  ~PythiaEventAccessor();
  void SetFileName();
  string GetFileName();
  void GetFile();
  int GetCMSEnergy();
  void GetParticlesOfEvent(vector<particleStore> &partVec, double V);
  int GetTotalEvents();
  void SetLastEvent();
  void GetCurrentEvent();
  int GetLastEvent();
};
```

From the file PythiaEventAccessor.cc:

```
/*
 * This is Robert's Pythia Event Class implementation.
 *
 * This is a second version where the class only
 * needs to read the CM energy and velocity from the
 * Glauber model N-N collision. The appropriate root
 * file will then be accessed based on rounding the
 * input energy to the nearest value (integer value)
 * that corresponds to a root file. The events in
 * the root file are then accessed.
```

```
 *
 * 2/12/2007 created
 */

#include "PythiaEventAccessor.h"

PythiaEventAccessor::PythiaEventAccessor(double e)
  //3/13/07: PythiaEventAccessor should take in CM Enrgy and CM Velocity
{
  CurrentEvent = 0;
  //cout << "Current Event: " << CurrentEvent << endl;

  cmsEnergy = (int)(e+0.5); //round energy e to nearest integer

  SetFileName();

  f = new TFile((const char*)filename.c_str(), "READ");

  tree = (TTree*)f->Get("pyttree");

  nevents = tree->GetEntries();

  particles = new TClonesArray("TMCParticle",100000);

  partBranch = tree->GetBranch("part");

  partBranch->SetAddress(&particles);

  id = tree->GetBranch("part.fKF");
  energy = tree->GetBranch("part.fEnergy");
  px = tree->GetBranch("part.fPx");
  py = tree->GetBranch("part.fPy");
  pz = tree->GetBranch("part.fPz");

}

PythiaEventAccessor::~PythiaEventAccessor()
{
  // do nothing
}


//File Selection-------------------------------------------------------
```

```
void PythiaEventAccessor::SetFileName()
{
  //Round the input energy to the nearest root file energy.
  if(cmsEnergy < 13) Energy = 10;
  for(int i = 1; i < 40; i++) {
    if (((cmsEnergy >= (13 + 5*(i-1))) && (cmsEnergy < (13 + 5*i)))) {
      Energy = 10 + 5*i;
      break;
    }
  }
  std::stringstream ss;
  std::string str;
  ss << Energy; //Convert integer Energy to string str
  ss >> str;
  //cout << str << endl;
  string namepart1 = "../rhic/generatedData/Kt=1.0/ppKt=1.0/pp25k";
  string namepart2 = "GeVKt10MSEL1Set700.root";
  //Construct file name:
  filename = namepart1 + str + namepart2;
  //cout << " The filename is: " << filename.c_str() << endl;

  //Convert filename to C-string:
  // const char* inrootfile = filename.c_str();
}

string PythiaEventAccessor::GetFileName()
{
  return filename;
}

int PythiaEventAccessor::GetCMSEnergy() //This function may not be needed
{
  return Energy;
}

void PythiaEventAccessor::GetParticlesOfEvent(vector<particleStore>
                                              &partVec, double V)

{
  double pxtemp, pytemp, pztemp, etemp, pztemp2, etemp2;
  int pid, nhits;

  tree->GetEvent(CurrentEvent);
```

```
  nhits = particles->GetEntries();

  leafid = id->FindLeaf("part.fKF");
  leafenergy = energy->FindLeaf("part.fEnergy");
  leafpx = px->FindLeaf("part.fPx");
  leafpy = py->FindLeaf("part.fPy");
  leafpz = pz->FindLeaf("part.fPz");


  for (int j = 0; j < nhits; j++) {
    pid = leafid->GetValue(j);
    //Lorentz transformation to lab frame performed here.
    //etemp = lab energy of produced particle
    etemp = (1/sqrt(1 - pow(V,2)))*
      (leafenergy->GetValue(j) + V*leafpz->GetValue(j));
    pxtemp = leafpx->GetValue(j);
    pytemp = leafpy->GetValue(j);
    //pztemp = lab pz of produced particle
    pztemp = (1/sqrt(1 - pow(V,2)))*
      (leafpz->GetValue(j) + V*leafenergy->GetValue(j));

    particleStore *pstore = new particleStore(pid, pxtemp, pytemp,
      pztemp, etemp);

    partVec.push_back(*pstore);
  }
  //cout << "Inside GetParticlesOfEvent: Vector size: " << partVec.size()
  //   << endl;

  CurrentEvent++;
}

int PythiaEventAccessor::GetTotalEvents()
{
  return nevents;
}
```

From the file headerAA.h:

```
// ================================================
//
// Make these objects of type TObject and make a set of
// Get and Put methods for all private data members
// Is this the best way?
```

```
//
//
// Robert Zaballa
// ==============================================

// by making this an object of type TObject I can use
// TClonesArray and then easily store things in a TTree
// then when you read the TTree you just need to include
// this header file and you can access the data by these
// get methods
//
// add ifdef-ing on this routine

#ifndef ROOT_TObject
#include <TObject.h>
#endif

class headerAA : public TObject {

 private:

  Int_t    evt;
  Int_t    nparticles; //Number of produced particles in AA event
  Int_t    npartTarg;
  Int_t    npartProj;
  Int_t    nNNcoll;
  Double_t impactparam;
  Double_t eccent;
  Double_t elossfraction;
  Double_t cme;
  Double_t v2;
  Double_t overlapArea;


 public:

  headerAA(){;}              // constructor
  virtual ~headerAA(){;}     // destructor

  Int_t    GetEvt()            {return evt;};  // just inline the get here
  Int_t    GetNparticles()     {return nparticles;};
  Int_t    GetNpartTarg()      {return npartTarg;};
  Int_t    GetNpartProj()      {return npartProj;};
  Int_t    GetNNcoll()         {return nNNcoll;};
```

```
  Double_t GetImpactParam()    {return impactparam;};
  Double_t GetEccent()         {return eccent;};
  Double_t GetELossfraction()  {return elossfraction;};
  Double_t GetCME()            {return cme;};
  Double_t GetV2()             {return v2;}; //for v2 of Participants
  Double_t GetOverlapArea()    {return overlapArea;};


  void SetEvt(Int_t Evt)                      {evt = Evt;};
  void SetNparticles(Int_t Nparticles)        {nparticles = Nparticles;};
  void SetNpartTarg(Int_t NpartTarg)          {npartTarg = NpartTarg;};
  void SetNpartProj(Int_t NpartProj)          {npartProj = NpartProj;};
  void SetNNcoll(Int_t Ncoll)                 {nNNcoll = Ncoll;};
  void SetImpactParam(Double_t ImpactParam)   {impactparam = ImpactParam;};
  void SetEccent(Double_t Eccent)             {eccent = Eccent;};
  void SetELossfraction(Double_t ELossfraction)
    {elossfraction = ELossfraction;};
  void SetCME(Double_t CME)                   {cme = CME;};
  void SetV2(Double_t V2)                     {v2 = V2;};
  void SetOverlapArea(Double_t OA)            {overlapArea = OA;};

  ClassDef(headerAA,2)

    };
```

From the file headerAA.C:

```
#include "headerAA.h"

// this is a macro
ClassImp(headerAA)
```