修 士 学 位 論 文

GEANT4による BESS-PolarII測定器シミュレーターの開発

2009年2月6日

物理学専攻　素粒子物理学研究室

077s111s

楠本 彬

神戸大学大学院理学研究科博士過程前期課程

目 次

表 目 次

図 目 次

概 要

BESS 実験（超伝導スペクトロメーターを用いた宇宙線観測気球飛翔実験）では、初期宇宙における素粒子現象の解明を目的として、主に低エネルギーの反陽子探索を行っている。2004 年に南極での 1 度目の実験「BESS-Polar 実験」が行われた。さらに、2007 年には 2 度の実験「BESS-PolarII 実験」が行われた。本研究で、BESS-PolarII 実験のためにデータ収集システムの FADC 接続、USB デバイスドライバー、通信システム、自動化処理などを開発・改良し安定した高速なシステムを構築することができた。現在、観測データは解析を行われている。

反陽子の Flux を求めるためには検出器に降り注いだ宇宙線のうち検出器がどの程度検出できるのかを評価しなければならない。そのため、BESS ではシミュレーションを用いてその割合を算出し、Flux を求めている。

BESS のシミュレーターは BESS-PolarI 実験までのものは GEANT3 で構築されたものがあるが、BESS-PolarII 実験のものは存在しない。そこで、本研究では GEANT4 を用いて新たにシミュレーターを開発した。

本研究でシミュレーターを GEANT4 で開発し、Cross Section などの物理を移植、PolarII の Geometry 入力、Acceptance 等でシミュレーションの結果を評価し、BESS-PolarII で反陽子解析に使う準備が整った。

本論文の構成は、第 1, 2 章で BESS 実験について説明し、第 3, 4 章で実験にあたり開発・改良したデータ収集システムについて説明する。その次の章からは、本研究のメインテーマであるシミュレーションについて説明する。第 5 章で BESS でのシミュレーションについて述べ、第 6 章で開発したシミュレーターの説明をして、第 7 章でシミュレーションの正当性を評価する。

第1章　Introduction

宇宙線反陽子の起源は、衝突起源のものが考えられているが、1 GeV 以下の低エネルギー領域に衝突起源モデルに対して若干過剰な分布が観測されている。これは、初期ブラックホール（PBH）などの宇宙起源のものではないかと予測されているため、BESS ではそのような低エネルギー領域の反陽子を観測している。

1.1　宇宙起源反陽子

反陽子の起源として、衝突起源のものの他に宇宙起源のものが考えられている。

衝突起源反陽子流束の理論モデルによるエネルギースペクトルは 2 GeV 付近にピークを持つと考えられる。しかし、1 GeV 以下の低エネルギーの領域に衝突起源反陽子では説明のできない分布の兆候が観られ、現在観測・解析が進められている。

これに対し、衝突起源以外では宇宙初期の素粒子現象を起源とする宇宙起源反陽子が考えられている。スティーブン・ホーキング（S.W.Hawking）は宇宙初期における素粒子現象として、原始ブラックホール（PBH : Primordial Black Hole）の生成を提案し、その蒸発に伴う陽子、反陽子の生成を予言した。宇宙初期に物質密度の揺らぎをきっかけとして、多数の原始ブラックホールが生成され、やがてエネルギーを放出しつつ質量は減少すると考えられる（ホーキング輻射)。生成時の質量が 10 億トン程度の原始ブラックホールは、現在の宇宙で寿命を迎え、爆発的にエネルギーを失い蒸発する時期を迎えていると予言している。原始ブラックホールの表面近傍において真空から仮想粒子対が生成され、かつ粒子がブラックホールに吸い込まれると、反粒子がブラックホール表面から放出される。その一部は反陽子として銀河の中を伝搬し、地球に降り注いでいる可能性がある。

宇宙線反陽子の流束は陽子に対し 10^{-4} 以下の極微量であり検出は非常に難しく、反陽子の 100 倍以上の流速がある電子は同じ負電荷をもち大きなバックグラウンドとなるため、観測器には大立体角かつ高い粒子選別能力が求められる。また、衝突起源反陽子のスペクトルは、太陽活動の 11 年周期の変化及び 22 年周期の磁極反転に伴いスペクトルに変調を受ける。より精密な衝突起源反陽子スペクトルの観測を行うためにはこの変化を正確に把握することが重要である。

1.2 反陽子観測の現状

1.2.1 過去の観測

宇宙線反陽子はゴールデン（Golden）らによって 1979 年に初めて観測が報告された。反陽子は陽子の反粒子なので、陽子と星間ガスの衝突が起源であると考えられる。しかし、1981 年にバフィントン（Buffington）らが衝突起源反陽子としての予測を大きく超える低エネルギー反陽子の観測結果を報告した。これにより、衝突起源以外の起源を探る様々な理論的考察、実験的検証が行われてきた。

1.2.2 BESS 実験による観測

BESS では、1993 年に行われた最初のフライトでは初めて「質量による粒子の同定」という確実な方法で宇宙線反陽子を 4 イベント確認した [1]。その後改良を加えながら低エネルギー宇宙線反陽子の測定を主な目標とし 9 回のフライトを行い、これまでに全体で 2000 イベント以上の宇宙線反陽子現象を確認している。1995+1997 年のデータの 1GeV 以下の領域で反陽子のフラックスが理論予測値よりやや平坦に見える (図 1.1)。

BESS-Polar 実験 [2] では BESS 実験での経験をもとに、BESS 実験の 10 倍以上の長時間観測と、さらに低エネルギーの反陽子を測定するために従来の BESS 測定器よりも低物質、軽量、低消費電力の測定器を開発し、2004 年 12 月南極において最初の長期間のフライトに成功した。さらに、2008 年 12 月には BESS-Polar 実験の経験を元に 2 度目の南極における実験、BESS-PolarII 実験が実施され、より高精度の実験結果が期待される。

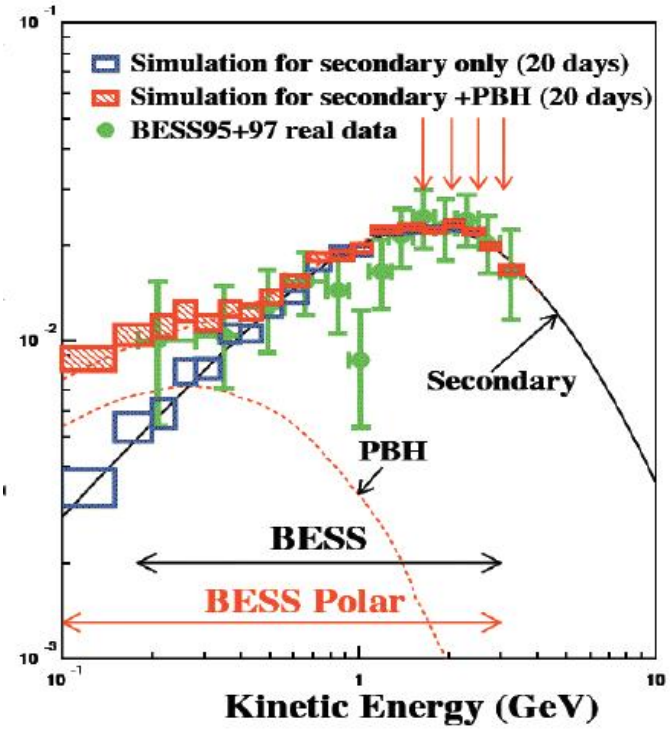

図 1.1: 南極における 20 日間のフライトを想定した、低エネルギー反陽子流束のエネルギー分布。太陽活動極小期に観測を行うことで、一次 (宇宙) 起源反陽子 (PBH を起因) による低エネルギー反陽子スペクトルの形の変化を観測することが可能となる。

第2章　BESS-Polar実験

超伝導スペクトロメーターを用いた宇宙線観測気球飛翔実験（BESS: Balloon-borne Experiment with a Superconducting Spectrometer）は、主に「初期宇宙における素粒子現象」の理解を目的として、高エネルギー加速器研究機構・東京大学・神戸大学・宇宙航空研究開発機構（ISAS/JAXA）・NASA・メリーランド大・デンバー大によって推進されてきた日米国際共同実験である [2]。

BESS-Polar 実験は、超伝導ソレノイドマグネット及び高精密かつ大立体角のスペクトロメーターを用いて、低エネルギー宇宙線が数多く降り注ぐ地球磁極に近い南極にて長時間観測を行う計画である。2004 年 12 月に約 9 日間をかけ南極周回飛翔実験に成功した（BESS-PolarI 実験）[3]。また、宇宙線反陽子の感度が最大となると考えられている太陽活動極小期は 2007〜2008 年と予想されておりこの時期に再び南極にて長期飛翔実験を行うことで統計的に精度の高い宇宙線反陽子の測定・探索が可能となる（BESS-PolarII 実験）。

2.1　BESS-Polar計画

BESS 実験では、0.1GeV から 10GeV までの低エネルギーの反陽子を観測・探索している。しかし、低エネルギーの荷電粒子は地球磁場の影響を非常に強く受ける。粒子は磁力線に巻きつくように運動するので、地球磁場が地面に対し水平になっている日本では観測することが困難である。そのため、磁場が鉛直方向を向いている極地方で観測する必要がある。2004 年 12 月には初の南極での観測、BESS-PolarI 実験が実施された。低エネルギー領域の統計を上げることに成功した。

さらに、低エネルギー宇宙線は地球磁場の他に太陽風の影響を強く受ける。そのため、太陽活動が弱い時期の方が観測に適している。太陽活動は 11 年周期で極大期と極小期が繰り替えされるので、2008 年はちょうど太陽活動極小期にあたる (図 2.1)。その時期に合わせて 2007 年 12 月に 2 度目の観測 BESS-PolarII 実験が実施された。

2 度の観測で一次起源反陽子の統計量を増やすと共に、極小期前に PolarI では二次起源の Flux を正確におさえ、極小期である PolarII では PolarI との結果を比較して低エネルギーの Flux の変化を見る。

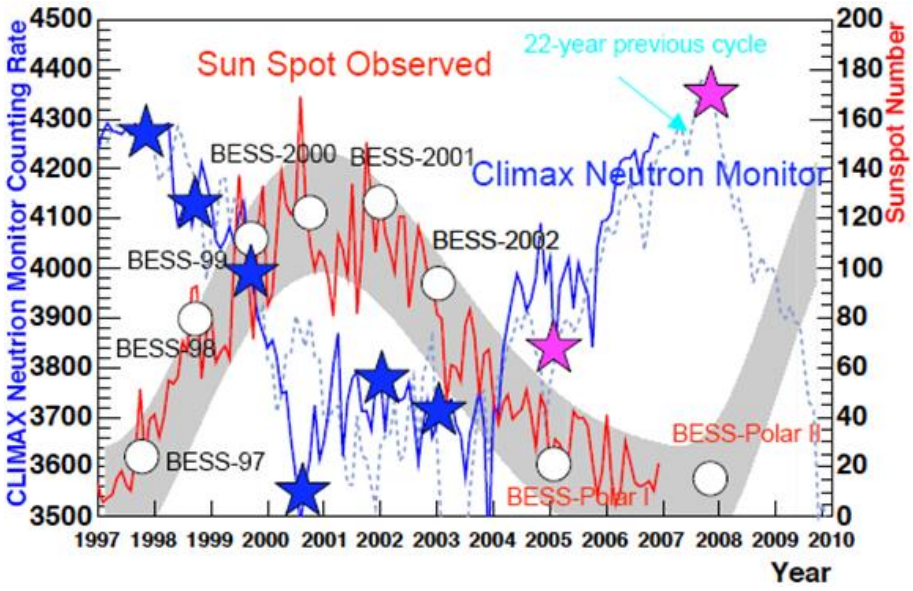

図 2.1: 太陽活の変化。BESS-PolarII は太陽活動極小期に合わせて 2008 年に実施された。

2.2 BESS-Polar 測定器

BESS 測定器は加速器・素粒子実験で培われてきた技術を基礎とし設計されており、強磁場空間を作り出す超伝導ソレノイドマグネット、中央飛跡検出器 (JET/IDC)、TOF カウンター (TOF)、エアロジェルチェレンコフカウンター (ACC)、高速データ収集システム (DAQ) によって構成された超伝導スペクトロメーターである。

粒子測定は各測定器を通過することで行われる。このため、BESS スペクトロメーターは超伝導ソレノイドマグネットの薄肉化により粒子透過性を高め、入射粒子との相互作用を最小限に抑えつつもコイル内部磁場の均一化が図られており、それまでの宇宙線検出器と比べ大きな立体角を実現した。1993 年よりカナダ北部等で実施した計 9 回の気球飛翔実験を通じ、他の宇宙線観測を大きく上回る高精度・大統計の測定結果を残してきた。

BESS-Polar 実験では南極でそれまでより長期間の観測を行うために、測定器全体を軽量化し、さらに低エネルギーの粒子の観測のために限界まで物質量を削減した (図 2.2)。それによって、2000 年の実験まではトリガーが発生するまでに粒子は 18 g/cm^2 の物質量を通過しなければならなかったのが、BESS-Polar では 10 g/cm^3 まで削減することに成功した。

さらに、BESS-Polar 実験からより低エネルギーの反陽子を捕らえるために、マグネットの下に Middle TOF(MTOF) を搭載した（MTOF の詳細は 2.4.2 で述べる）ので、最小 5 g/cm^2 通過できればトリガーを生成することができる。BESS-Polar 測定器の全体図を図 2.3 に示す。

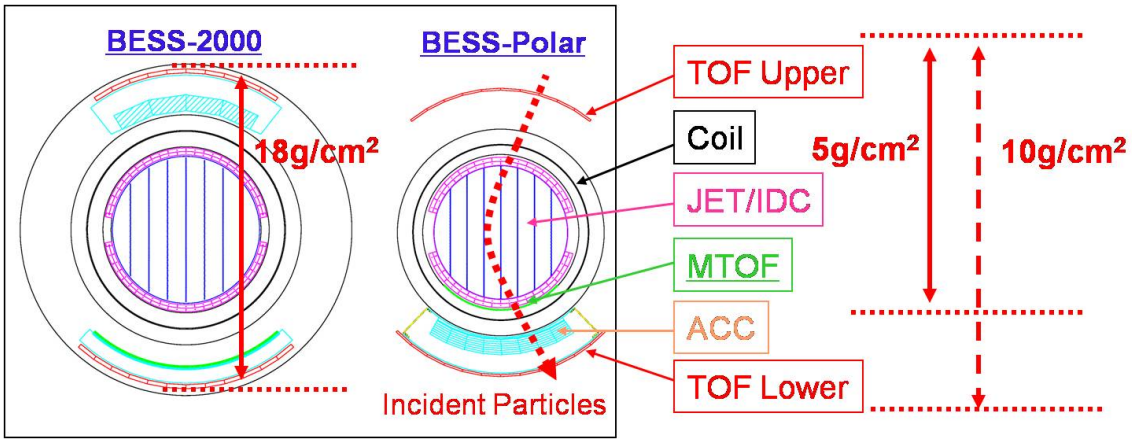

図 2.2: BESS 測定器と BESS-Polar 測定器

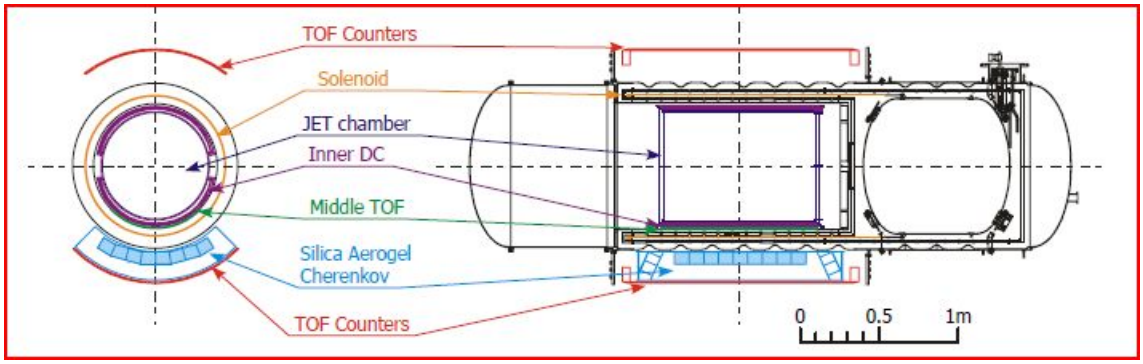

図 2.3: BESS-Polar 測定器断面図

2.3 測定原理

BESS-Polar 測定器では BESS 測定器と同様に粒子の質量を同定することにより粒子の識別を行っている。粒子の速度と運動量の間には関係式

$$\beta = \frac{pc}{\left(p^2c^2 + m^2c^4\right)^{1/2}} \tag{2.1}$$

（β：速さ/光速、p：運動量、c：光速、m：質量）が成り立つので、粒子の速度と運動量を測定することで質量を決定できる。磁場中を荷電粒子が通過するとローレンツ力が働き、その粒子は半径 r cm

$$r = \frac{p}{ZeB} \tag{2.2}$$

(Ze：電荷、B：磁場の強さ) の軌跡を描く。BESS 測定器には超伝導ソレノイドが搭載されており、内部には約 0.8 T の均一磁場がかけられている。これにより入射荷電粒子を曲げ、JET チェンバーと IDC（2.4.1 参照）において飛跡の測定がなされる。また、最外層には TOF、測定器中間下層には Middle-TOF が配置されていて、粒子の速度を測定している。さらに JET チェンバーと TOF カウンターの dE/dx から粒子の電荷の大きさ (Ze) がわかるので、Rigidity ($R = pc/Ze$) を使って、質量は以下の式で表される。

$$m^2 = R^2Ze^2\left(\frac{1}{\beta^2} - 1\right) \tag{2.3}$$

以上より、縦軸に $1/\beta$、横軸に Rigidity (GV) をとって測定値を xy 平面にプロットすると、図 2.4 のように質量をパラメーターとしてバンドを形成する。BESS 実験では測定器に磁場がかけられているので、粒子の軌跡の曲がる方向から電荷の正負が識別できる。そのため、粒子判別バンドの陽子部分を負電荷側に適用することで反陽子も同定することができる。

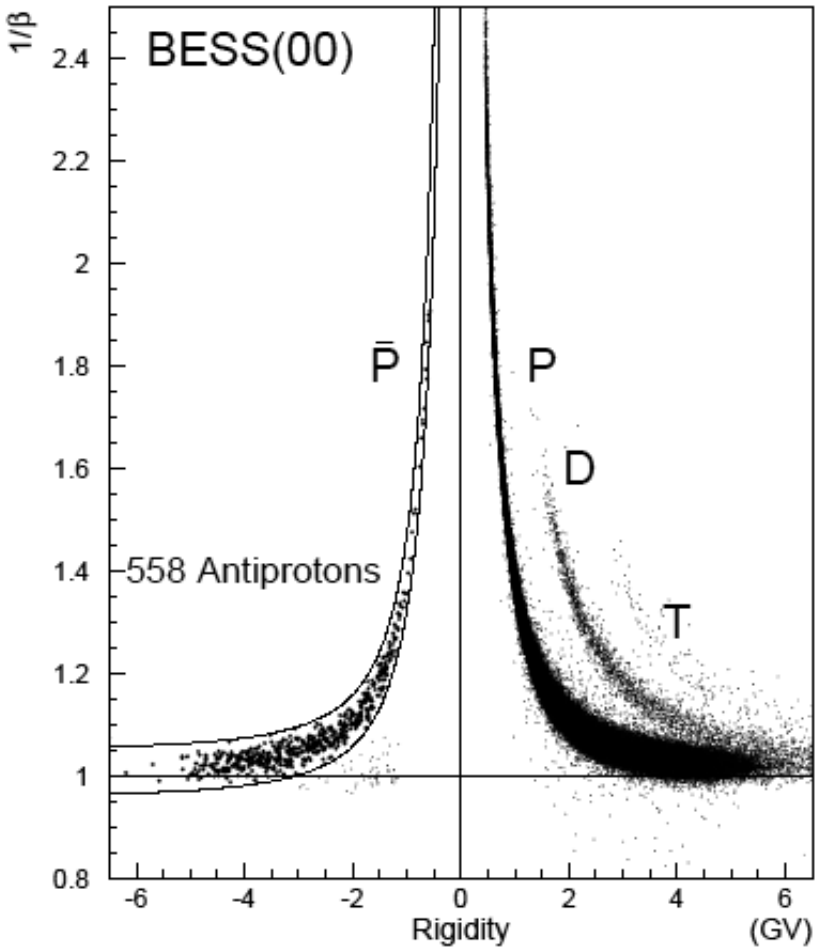

図 2.4: β^{-1} vs Reigidity プロット

2.4 BESS-PolarII 測定器

2.4.1 中央飛跡検出器（JET/IDC）

ソレノイドコイル内部には、円形型のジェット型ドリフトチェンバー (JET) と、円弧状の Inner Drift Chamber(IDC) が配置されており、磁場で曲げられた粒子の軌跡を追跡する。(図 2.5, 図 2.6)

粒子の軌跡を最大 48(JET)+4(IDC) 点で測定し、magnetic rigidity を求めている。長期観測を通してガスクオリティを維持するためのフローシステムの開発と CO_2 とアルゴンの混合ガスから CO_2 のみに変更したことによるガスゲイン低下を補う為の高ゲインアンプの開発が行われた。

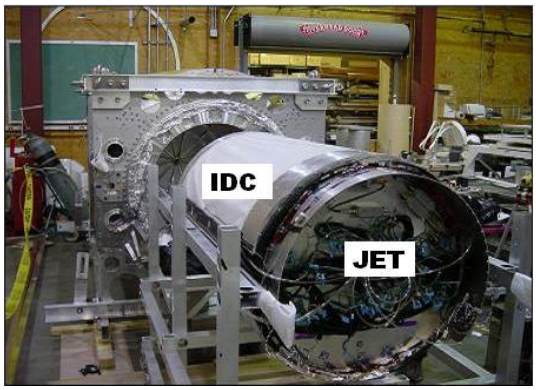

図 2.5: JET・IDC

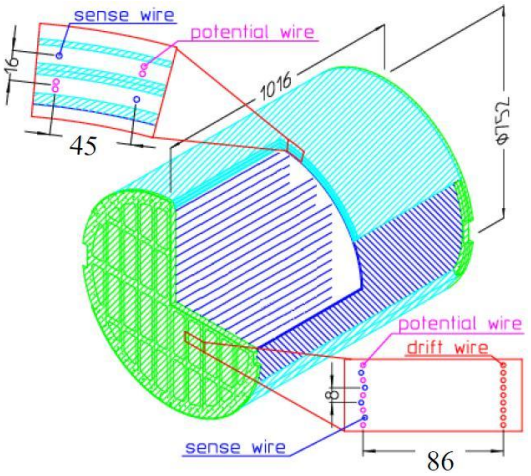

図 2.6: JET・IDC 設計図

2.4.2 TOF Counter

宇宙線の粒子の飛行時間とエネルギー損失の測定を行うため、測定器の上下に TOF カウンターが配置されている（図 2.7, 2.8）。厚さ 1.2 cm のプラスチックシンチレーターからの信号はライトガイドを介し、両側に取り付けられている PMT(Photo Multiplier Tube) により読みだす。

中央飛跡検出器とマグネットクライオスタット内壁下部との間には、厚さ 5.6 mm のプラスチックシンチレータと PMT からなる MiddleTOF(MTOF) を搭載している（図 2.11）。粒子通過領域の物質量を増やさないように、光ファイバーはタンクの壁面を通り PMT に接続している（図 2.9）。MTOF によりマグネット下部を通過することができない低エネルギー粒子を観測することが可能となった。PolarI における MTOF は片側読み出しであり、時間分解能の位置依存性・トリガータイミングに不定性があっが、PolarII では粒子通過領域を回避しながら、ヘリウムタンク側の信号の読み出しに成功し両側読み出しの MTOF を実現した（図 2.10）。

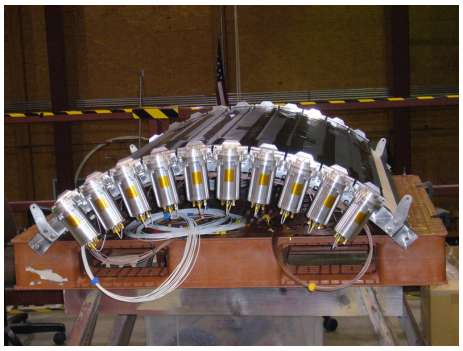

図 2.7: 検出器最上部に設置されている TOF カウンター (UTOF)

図 2.8: 検出器最下層に設置されている TOF カウンター (LTOF)

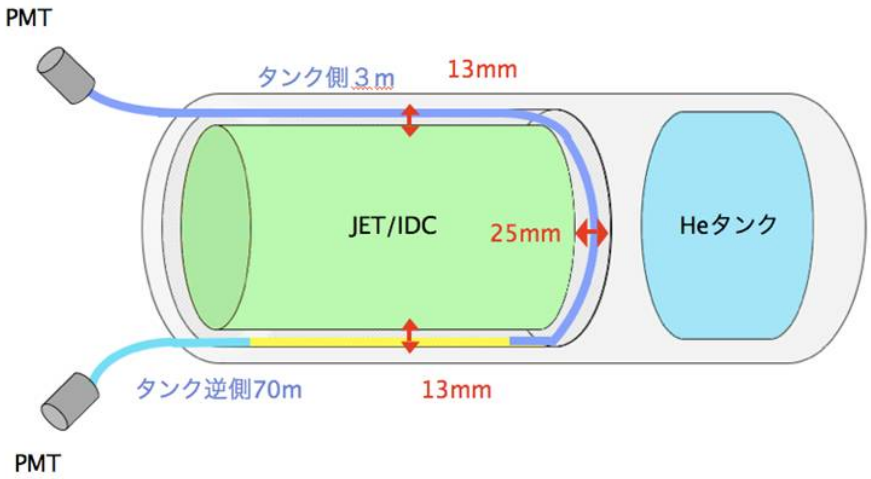

図 2.9: MTOF 断面図

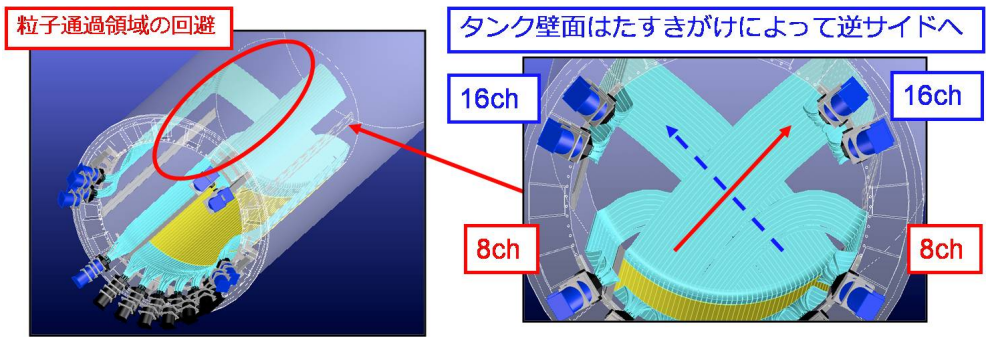

図 2.10: MTOF の構造

図 2.11: MTOF をインストールした様子

2.4.3 Aerogel Cherencov Counter

クライオスタットと下部TOF カウンターとの間には、Aerogel Cherenkov Counter(ACC) が配置されている。BESS-Polar 実験における ACC は閾値型測定器で、反陽子のバックグラウンドとなる軽い粒子 (e, μ) を分離する際に使用する。PolarI での ACC は当時の製作可能形状 (大きさ) の制約から、以前より小型に設計された。この変更により光量が以前より失われる結果となった。PolarII では ACC ブロック、構造体の見直しがなされ PolarI 以前の ACC と同等の性能となっている (図 2.12, 2.13)。表 2.1 に PolarI と PolarII での ACC パラメータの比較を示す。

表 2.1: ACC パラメータ

	PolarI	PolarII
ブロックサイズ	100 × 100 × 10mm	190 × 280 × 20mm
屈折率	1.02	1.03
識別領域	～4.6 GeV	～3.8 GeV
光量比	1	1.47

2.4.4 超伝導マグネット

測定器の中央部分には、直径 0.8 m・長さ 1.0 m の超伝導ソレノイド型マグネットが組み込まれている（図 2.14)。超伝導マグネットにより、検出器領域には約 0.8 T の強磁場空間が作り出される。BESS-Polar ではマグネットの薄肉化が図られており、コイルとクライオスタットを含めた物質量は、以前の BESS マグネットの 4.2 g/cm^2 に対し、2.5 g/cm^2 まで削減されており、より低エネルギー粒子の観測が可能になった。また、磁場の一様性は十分に高く、分布は 9%以下である。

図 2.12: Aerogel Block

図 2.13: Aerogel Cherenkov Counter

超伝導ソレノイドコイルはPolarIと同様の設計であるが、PolarIIでは南極２周回(約20日)のフライトが目標であり、フライト時間を直接制限している液体ヘリウムのライフを延ばすため、マグネットのタンク・クライオスタットの改良を行った。

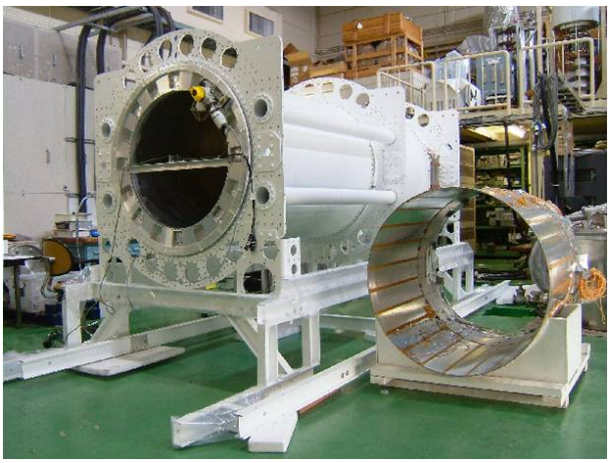

図 2.14: 超伝導マグネット

2.4.5 Solar パネル

従来のBESS実験では１次電池により電力を供給していたが、BESS-Polar実験では長期フライトを見込んでいるため新たに太陽電池システムによる電力供給方法が採用されている。PolarIでは必要発電量450Wに対して、発電量900Wの余裕を持った設計であった。PolarIIではシステムの見直しがなされ、コンパクト(4段から3段に変更)な設計となった（図2.15)。これにより太陽電池システムの総重量を50kg削減し、また打ち上げ時の安全性が増している。

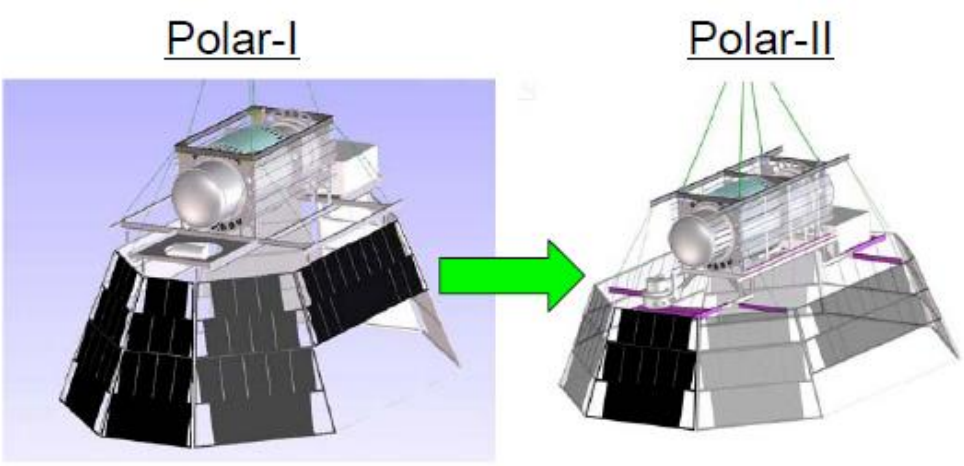

図 2.15: 太陽電池システムのコンパクト化

2.4.6 データ収集システム

上 TOF と下 TOF もしくは MTOF のコインシデンスによりトリガーを出力し、各読み出しモジュールが、対応した検出器からデータを読み出す。QDC,TDC,Trigger の 3 つのデータは MU2 ボードで一つにパックされ USB 信号に変換され、FADC から送られてくる JET のデータとまとめて HDD に記録される（図 2.16)。1Tbyte の HDD16 個をデータ記録に使用し、すべてのイベントをセレクション無しに記録することが可能である。

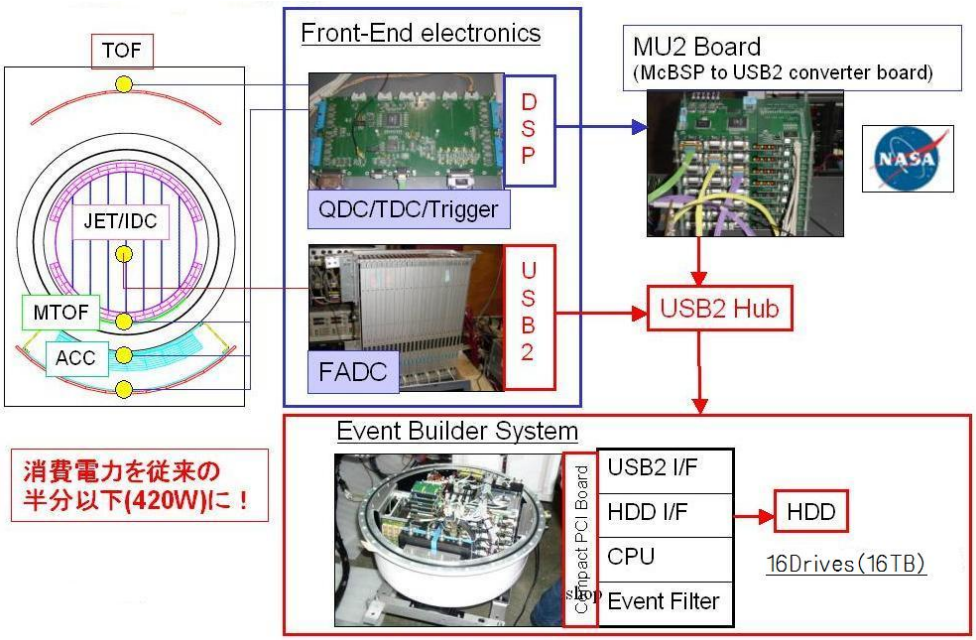

図 2.16: BESS-Polar データ収集システム

2.5 BESS-PolarI フライト

BESS-PolarI 実験は 2004 年に南極で実施された。2004 年 12 月 13 日 18 時 54 分 (ニュージーランド時間) に Williams Field より打ち上げられ、フライトは約 8.5 日間の行われ、2.1 TB、9 億イベントの収集に成功した。図 2.17 は Williams Field での打ち上げの様子、図 2.18 は飛行経路である。

PolarI のフライトでは打ち上げ直後に TOF の PMT が次々に放電し、半数近くの PMT が使用不可能になるという問題が起こった。TOF はシンチレーターの両サイドに PMT を設置し両読み出しでトリガーを発生させる予定だったが、問題が起こった PMT は HV を落として片側の PMT のみでトリガーを出すようにして対処した。原因についてはっきりしたことは解明されてないが、低温テスト、低圧テストはされていたが、低温低圧テストは行われていなかったので、その様な環境下で何か問題が起こったのではないかと推測されている。

BESS-PolarII では同様の問題が起こらないように非常に丁寧に低温低圧テストが行われた [4]。

BESS-PolarI の解析は終了し、BESS-PolarI のデータからは反陽子の Flux は二次起源の Flux と一致している、という結果が得られた [5]。図 2.19 に BESS-PolarI の解析で得られた反陽子の Flux, 図 2.20 に陽子反陽子比を示す。BESS-PolarII では統計を上げてより詳しい解析を行う。

図 2.17: BESS-PolarI 打ち上げの様子

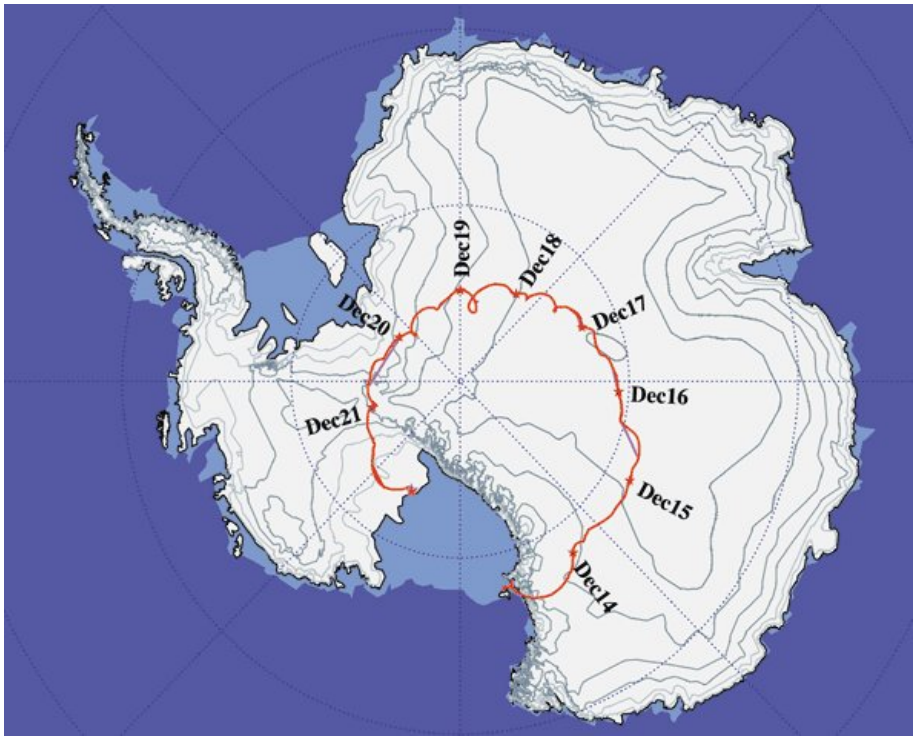

図 2.18: BESS-PolarI フライトの軌跡

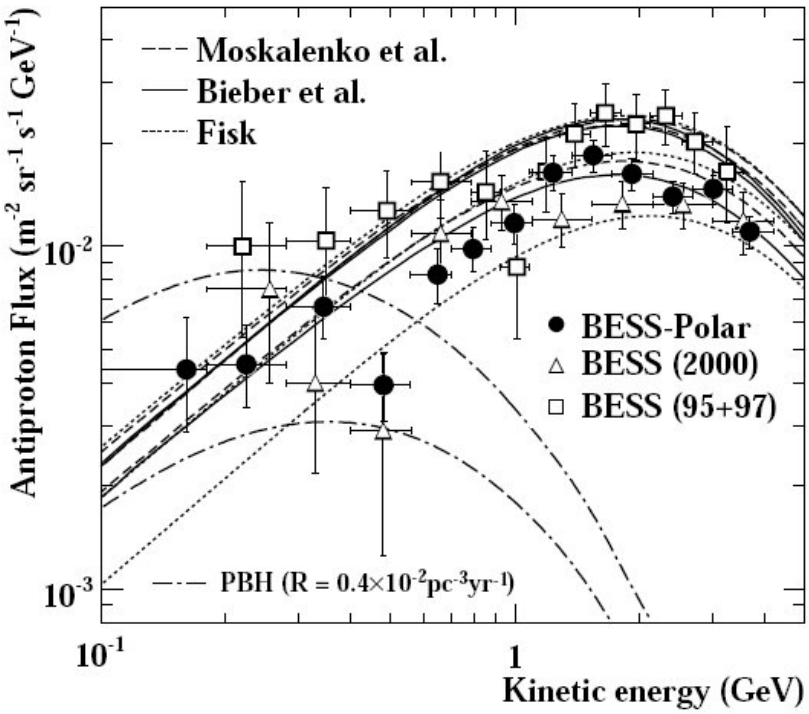

図 2.19: BESS-PolarI の観測データから得られた反陽子の Flux

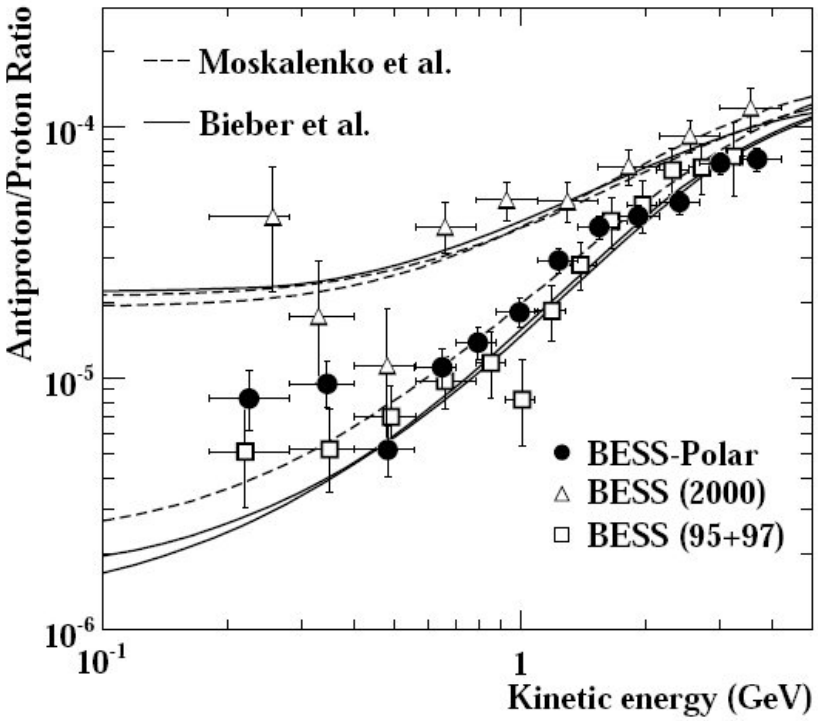

図 2.20: BESS-PolarI の観測データから得られた陽子・反陽子の比率

2.6 BESS-PolarII フライト

BESS-PolarII スタッフは 2006 年頃から前回の実験の反省を元に、KEK を中心に測定器の開発・改良を進めた。表 2.2 に BESS-PolarI 実験から BESS-PolarII 実験へ変更された点をまとめる。

表 2.2: BESS-PolarI から BESS-PolarII への測定器の変更点

項目	BESS-PolarI 実験	BESS-PolarII 実験
超伝導マグネットの寿命	11 days	22 days
TOF-PMT の低温低圧環境対策	ポッティング	アルミニウム製機密容器
MTOF 読み出し	片側読み出し	両側読み出し
ACC の性能（Rejection Power）	630	1000〜5000
太陽電池システム	4 段構造 900 W	3 段構造 450 W
Data Storage	3.6 TB	16 TB
Acceptance	0.17 m^2sr	0.27 m^2sr

2007 年 1 月にはアメリカ・メリーランド州にある GSFC (= Goddard Space Flight Center) に移動し測定器の建設を行った（図 2.21)。

同年 7 月には全ての測定器を作り終え、テキサス州パレスティンの CSBF(= Columbia Scientific Balloon Facility) へ移動し、CSBF の通信システムとの動作テスト・噛み合わせ試験を行い、BESS 測定器（および、ソーラーパネル）が正しく吊り上げられ、DAQ システムが磁場中で正常に動作し、通信システムが連携が正しく機能するかを確認した（図 2.22)。

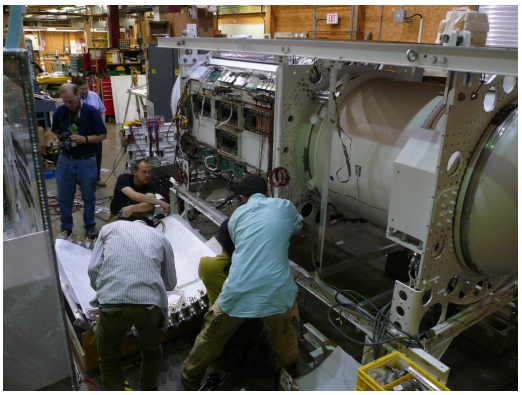

図 2.21: GSFC での作業の様子。写真は ACC をインストールしているところ

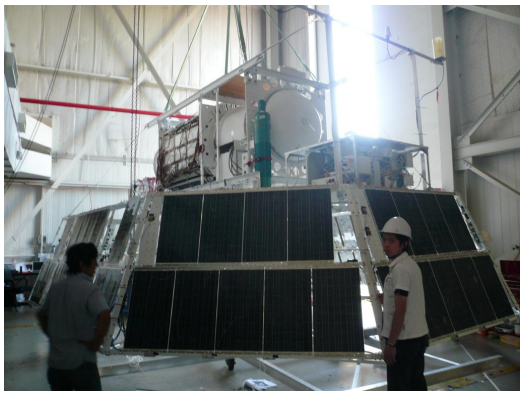

図 2.22: CSBF での噛み合わせ試験の様子

全ての準備を終えて、スタッフは 2007 年 10 月 26 日に南極 McMurdo 基地入りをした。約 2ヶ月間の準備を経て 12 月に Compatibility Test に合格したのち、2007 年 12 月 22 日 17:30(UTC) に南極 McMurdo 基地近くの Williams Field から打ち上げら

れた (図 2.23))。打ち上げ直後に DAQ が不安定になったり、TOF PMT の信号のノイズが増えるなどの問題が発生したが、一度測定器の電源を入れ直すことで、DAQ の状態は改善し、安定した動作を続けた [6]。

2008 年 1 月 4 日には新たな太陽活動サイクルの黒点が観測されるというニュースが報道され、また、コロナホールからの高速太陽活動の影響により、BESS のトリガーレートが太陽風の速度、ニュートロンモニターと同期して上下するという現象も確認された。

打ち上げ後、18 日間で南極上空を 1 周し、24.5 日後に液体 He、および、HDD の残量がなくったので、消磁を行い観測を終了した。その後、キャリブレーションを目的とした磁場なしのデータを約 1 日間取り、TOF PMT HV と DAQ 以外の電源を落として宇宙線のトリガーレートの観測のみを行いながら、気球の切り離しに備えた。

日が経過するにつれ、気球の軌跡が不安定になり、進路の予測が困難になったため、2008 年 1 月 21 日に気球を切り離し、南極点からおよそ 686 km 離れた西南極氷棚の上に着地、約 29.5 日間のフライトに成功した。（図 2.24）

図 2.23: BESS-PolarII フライト打ち上げの様子。2007 年 12 月 23 日に Williams Field で打ち上げ

パラシュートと測定器の切り離しはほぼ完璧で、測定器は上を向いた状態でそのまま着地した。着地後も通信機器は動作を続け、データを基地に送り続けた。

フライト終了が南極のシーズン終盤であったため、測定器の回収は翌年度以降に見送られたが、観測データの入った HDD が格納されている Data Vessel は無事回収された（図 2.25）。

BESS-PolarI と BESS-PolarII のフライトステータスの比較を表 2.3 に示す。今回の観測は太陽活動極小期であることと DAQ を改良したことによりトリガーレート

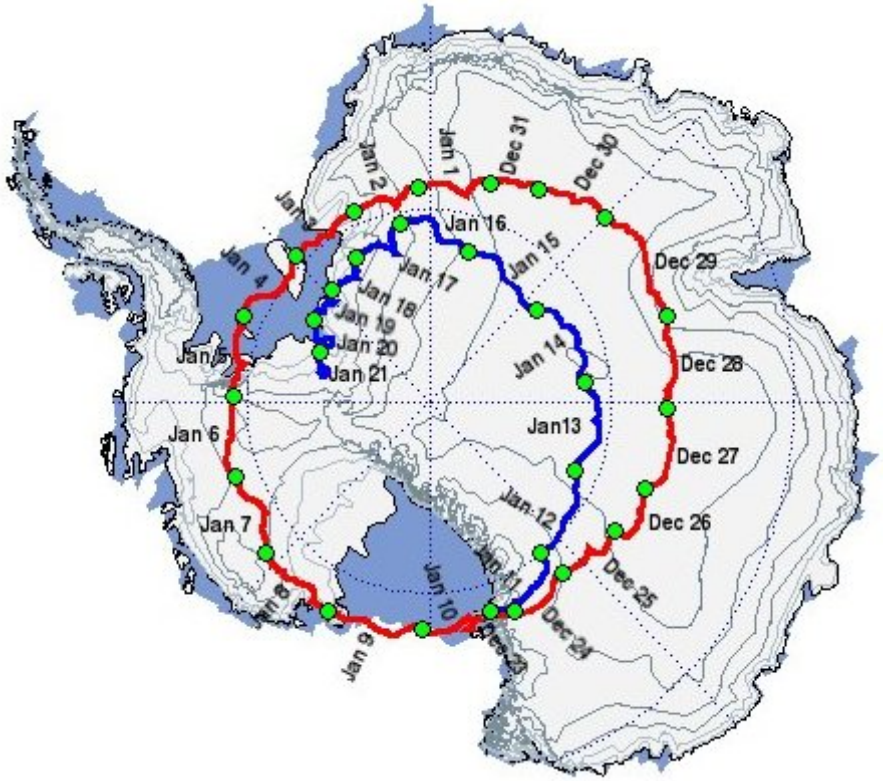

図 2.24: BESS-PolarII フライトの気球の軌跡。2008 年 1 月 21 日にカットダウン

図 2.25: Data Vessel を回収した様子

表 2.3: BESS-PolarI と BESS-PolarII のフライトステータスの比較

	BESS-PolarI (2004 年)	BESS-PolarII (2007 年)
Total Floating Time	8.5 days	29.5 days
Observation Time	8.5 days	24.5 days
Recorded Event	900 Million	4700 Million
Recorded Data Size	2.1 TB	13.5 TB
Trigger Rate	1.4 kHz	3.4 kHz
Live Time Fraction	0.8	0.82
Altitude	37〜39 km	34〜38 km
Redisual Air Pressure	4〜5 g/cm^2	4.5〜8 g/cm^2

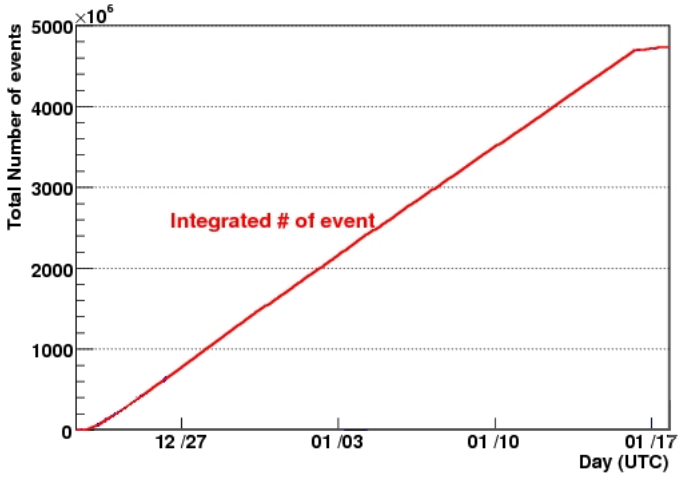

図 2.26: 収集したイベント数の推移。フライト期間中安定してデータを取り続けられたことが分かる。

が 3.4 kHz に上昇した。気球の高度が前回ほどあがらなかったために、残留大気は最大で 8 g/cm^2 になってしまった。しかし、観測時間も 24.5 日間に増えたので 13.5 TB, 47 億イベント収集することに成功した（図 2.26)。

回収したデータは現在解析が進められており、各検出器は期待通りの性能が出ていたことが確認されている。今回の解析で低エネルギー領域での 1 次起源の反陽子の分布に決着がつくことが期待されている。図 2.28 は現段階で得られている ID-Plot のデータである。

また、フライト中には太陽活動の変化を見ることもできた。図 2.27 上は Neutron Monitor で下は BESS のトリガーレートである。太陽活動の変化にそってトリガーレートも変化するという現象を確認することができた。

次章から 2 章かけて BESS-PolarII 実験のために開発・改良した DAQ システムについて述べ、第 5 章からは BESS-PolarII のシミュレーションについて述べる。

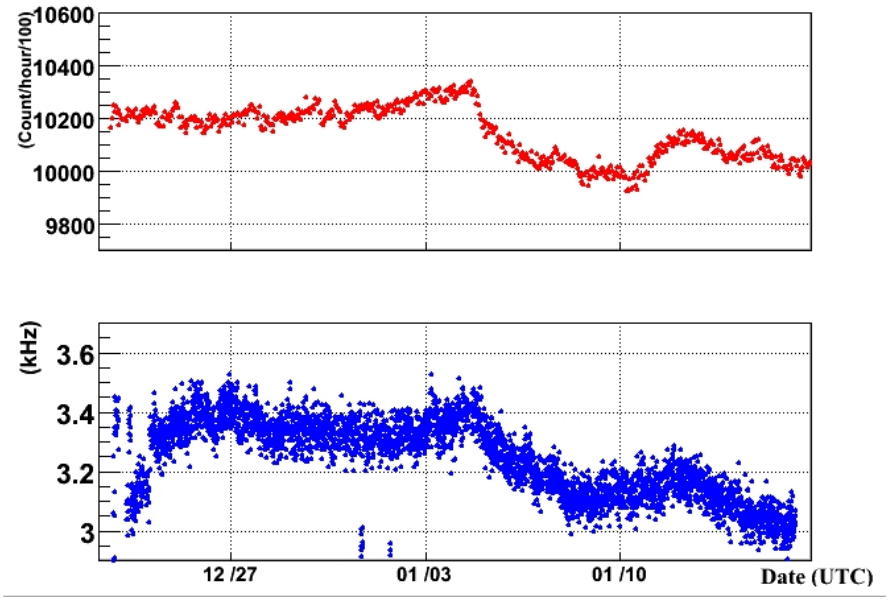

図 2.27: Neutron Monitor（上）と Trigger Rate（下）。太陽活動の変化に合わせて、Trigger Rate が変化している

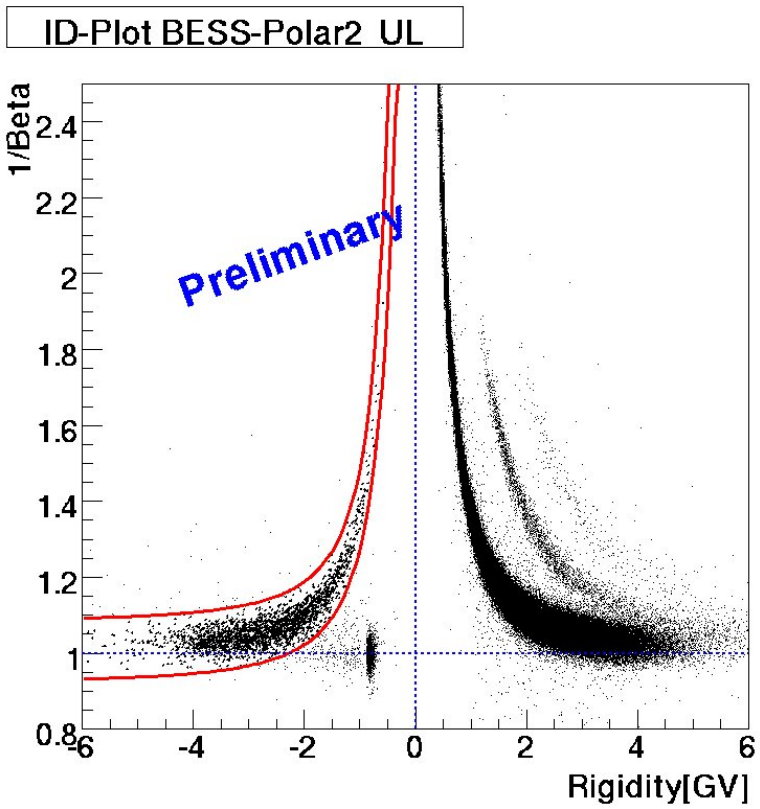

図 2.28: BESS-PolarII フライトデータから得た ID Plot（解析中）

第3章　BESS-PolarIIデータ収集システム

データ収集システム（以下、DAQ システム）とは、各検出器から収集したデータをイベントデータとして取りまとめ、HDD に保存するシステムである。BESS 実験は気球実験なので、実験中は DAQ コンピューターを直接操作することはできない。そのため、地上とのコマンド・モニターの通信も必要である。また、DAQ のシステムにも気球実験特有の制限を受ける。

この章では、BESS 実験で使用している DAQ システムについて述べる。

3.1　DAQ システムの概要

3.1.1　イベントビルド

BESS の DAQ プログラムは、検出器からデータが来ていないかを常に監視している。粒子が検出器を通過し、各機器からデータが送られると、DAQ プログラムはデータを取りまとめる（イベントビルド）。BESS のイベントデータ構築の概念図を図 3.1 に示す。

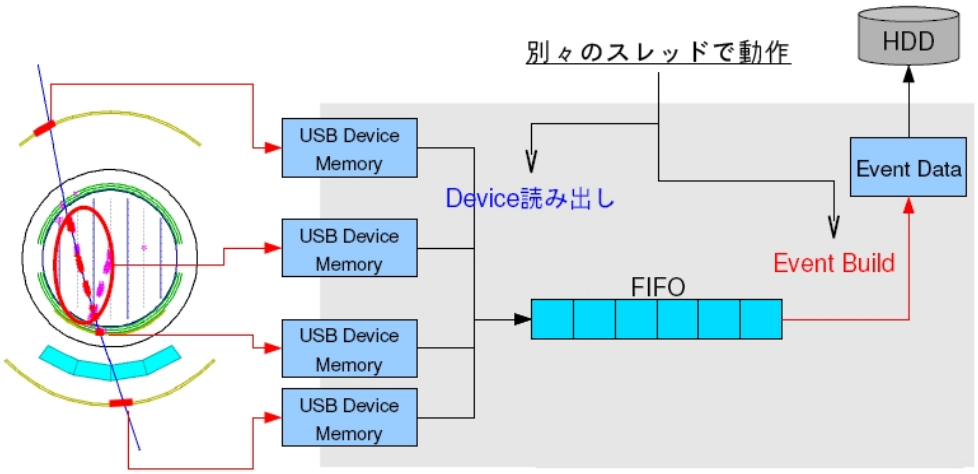

図 3.1: イベントビルドの概念図

BESS の検出器を粒子が通過すると、TOF の信号が TDC に入り、Trigger Board へ渡される。Trigger Board では、U,M,LTOF のどれから信号が入ったかを検知して

おり、UTOF+MTOF、もしくは、UTOF+LTOF の組み合わせであればトリガー信号を送出する。Trigger が発生すると、各エレクトロニクスがサンプリングを始め、JET/IDC の信号は FADC, TOF, ACC の信号は QDC によって取得する。これらの機器は全て USB で（TDC, QDC は MU2 ボードを介して）cPCI(= Compact PCI) に接続されている。

各検出器のデータは非同期的に cPCI に送られる。そのため、DAQ プログラムでは常に USB デバイスを監視しなければならない。BESS の DAQ プログラムは複数のスレッドに分かれて動作しており、USB デバイスからパケットを読み出すためのスレッド「DevReader スレッド」、イベントビルドを行うスレッド「EventBuild スレッド」、とわかれている。

DevReader スレッドでは、USB デバイスにパケットがあるかどうかを常に監視している。USB デバイスにパケットがあった場合は、即座に読み出しスタックに格納する。

EventBuild スレッドでは、FIFO を監視し、パケット内のイベント ID を元にそのイベントの全ての FADC からパケットが揃っているかをチェックする。イベント ID は FADC によってパケット内に埋め込まれる。全 FADC からのパケットが揃っていた場合、イベントデータが揃ったとみなしイベントビルドが始まる。FIFO から該当イベントのパケットを取り出し、1 つのイベントパケットとしてまとめて HDD に保存する。

BESS の DAQ システムでは、南極上空での高イベントレートに耐えなければならないので、デッドタイムを増やさないように、余計な処理は行わず、できうる限り高速化されている。そのため、イベントの選別は行わず、取得したイベントは全てそのまま保存している。

なお、イベントデータは基本的に全て HDD に保存されるが、地上でのモニタリングのために、一定間隔ごとに一部地上に送信している。

3.1.2 モニターシステム

BESS のモニターシステムは主に LON（=Local Operation Network）とよばれるネットワークシステムを用いて構築されている (図 3.2)。このシステムを用いて、HV の電圧・電流、各種温度モニターなどをモニターする。

LON とは、OSI 参照モデルの 7 層全てをサポートしている LON Talk プロトコルを元に作り上げられた分散型ネットワークシステムである。ノードと呼ばれるハードウェアがネットワーク上にあり、このノードがお互いに通信して各機器を制御する。

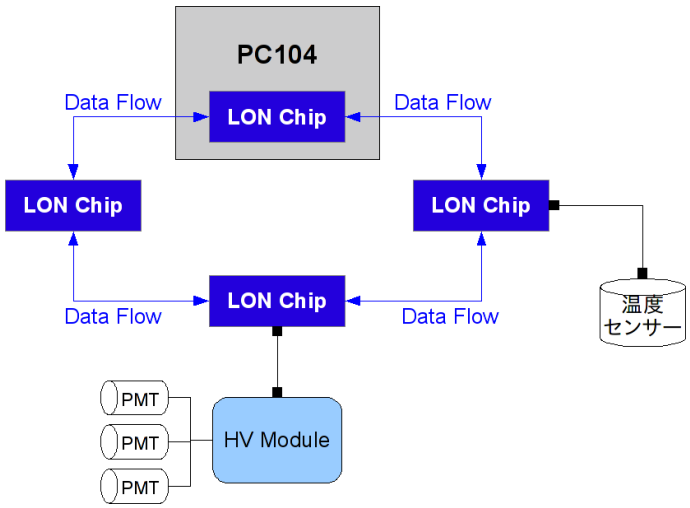

図 3.2: LON 構成図

3.1.3 通信システム

気球に搭載した DAQ システムと地上との通信には、LOS（=Line Of Sight)、および衛星通信（TDRSS, IRIDIUM）を用いる。表 3.1 に各通信システムの通信速度、図 3.3 に概念図を示す。これらの通信システムおよび、気球の技術は全て NFS (=National Science Foundation(アメリカ国立科学財団)) に協力を依頼している。

表 3.1: 各通信システムの通信速度

通信システム	LOS	TDRSS	IRIDIUM
通信速度	83.3 kbps	6 kbps	256 Byte/15 分

- LOS(=Line Of Sight)

  LOS は気球と南極の基地と直接通信を行う通信方法で、83.3kbps と最も速い通信方法である。しかし、直接通信を行うので、気球が基地の近くになければ通信できない。LOS が使用できるのは、打ち上げから大体 24 時間くらいである。ある程度距離が離れてくるとノイズが多くなり、まともにオペレーションできなくなってくるので、他の通信方法に切り替える。

  今回の実験では南極を約 1 周半したので、気球が再度基地に近づいた時には、再び LOS を使うことができた。

- TDRSS

  TDRSS(=Tracking and Data Relay Satellite System) とは、複数の衛星を用いた衛星通信ネットワークで、通信速度は 6kbps と LOS には劣るが、衛星を介して通信をするので、南極のほぼ全領域で通信することが可能である。フライトオペレーションでは、この通信方法を最も使用した。

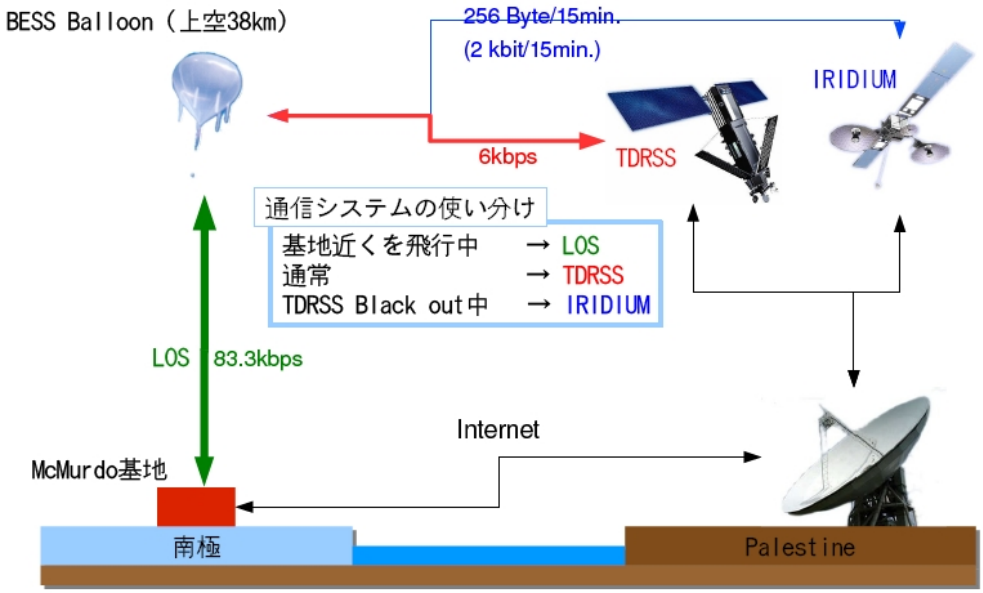

図 3.3: 通信システムの概念図

衛星通信の場合は、気球からのデータは衛星を介して、パレスティンにある NSF に送られる。NSF からはインターネット回線を通じて南極の基地まで転送される。南極からのコマンドはこの逆の経路をたどる。

ただし、TDRSS 通信の欠点は Black out という通信できない時間帯が存在することである。気球と TDRSS 衛星との位置関係によっては、地球の影に隠れて、気球からどの TDRSS 衛星も見えなくなることがある。この様な時は、通信することができない。

- IRIDIUM

  IRIDIUM も TDRSS と同様衛星通信である。TDRSS との違いは、衛星の数が全部で 66 基（当初の計画では 77 基になる予定だった）と非常に多いので、常にいずれかの衛星が見えているために、Black out する時間帯が存在しない。

  しかし、この通信方法の欠点は、15 分に 1 回 256 Byte 送るだけという非常に低速なものであるということだ。15 分おきに、その時たまたま届いたパケットのみを送信し、それ以外のパケットは破棄される。そのため、通常のフライトオペレーションを行うことはできず、TDRSS が Black out しているときのみ、わずかなコマンド・モニタリングに使用した。

3.2 システム構成

システム全体の構成図を図 3.4 に示す。

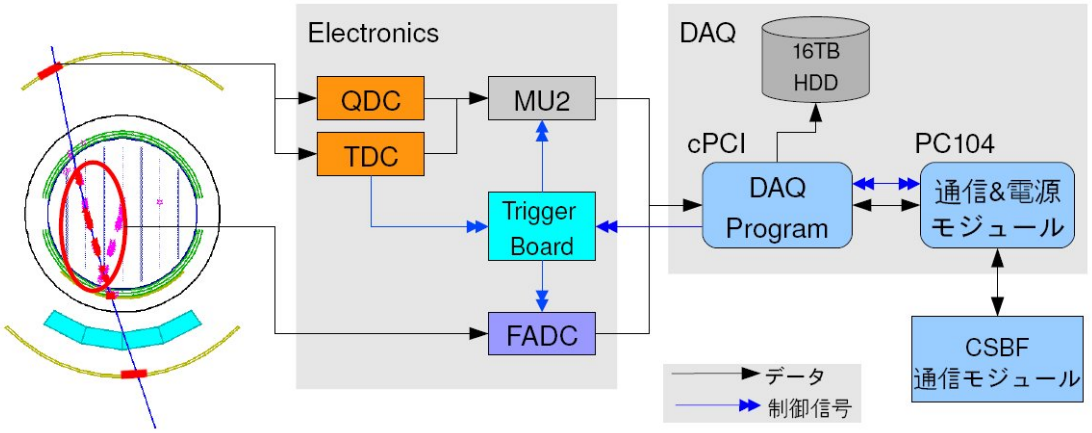

図 3.4: DAQ システムの構成図

BESS の DAQ システムはメインの DAQ コンピューターである cPCI(=Compact PCI)、通信・電源管理用コンピュータ PC104、および、観測データを保存する HDD からなる。

エレクトロニクス部は TOF, ACC の信号を ADC 変換する TDC, QDC とそれらと PC をつなぐ MU2 ボード、JET/IDC の信号を ADC 変換する FADC、TDC の信号を元にトリガーを生成する Trigger Board、および、クロックを発生させる Clock Generator からなる。

粒子が検出器を通過すると PMT からの信号が TDC で ADC 変換される。TDC の信号が Trigger Board に入り、UTOF+MTOF、もしくは、UTOF+LTOF の組み合わせで信号が出ていれば、Trigger Board がトリガーを発生させる。トリガーは、MU2 および FADC に送られ、MU2 は TDC（TOF）および QDC（TOF,ACC)、FADC は JET/IDC の信号を USB を介して cPCI に送られる。

cPCI では DAQ プログラムが各検出器からのデータを取りまとめ（イベントビルド)、HDD に保存すると同時に、モニターデータを PC104 に送信する。PC104 は地上との仲介役を担っており、CSBF の通信モジュールを利用して、地上と cPCI 間のデータ転送を行う。

3.2.1 Compact PCI (cPCI)

DAQ システムのメインコンピュータには cPCI（図 3.5）を使用している。気球実験では、コンピューターも気球に搭載するので、通常のデスクトップコンピューターは使用できない。できるだけ体積・質量を減らさなければならないので、cPCI を採用した。

表 3.2: BESS-Polar と BESS-PolarII で使用した CPU の性能比較

	CP306 (BESS-Polar)	CP307 (BESS-PolarII)
CPU	Intel Pentium M 1.4 GHz	Intel Dual Core Processor 1.66 GHz
DRAM	1G with ECC soldered	Max 4GB (2GB soldered + 2GB via SO-DIMM socket)
Chipset	Intel 855GME and ICH4	Intel 945GM and ICH7R
CPU L2 Cache	2MB	2MB
DRAM Speed	333 MHz	667 MHz
Ethernet	1x 1000Base-Tx 1x 100Base-Tx	2x 1000Base-Tx
Power Comsumption	13W	23W

BESS-PolarII では、BESS-Polar 実験より処理能力が高く、Dual Core の kontron 製の CPU「CP307」を採用し、DAQ の処理能力向上を図った。DAQ は、スレッド技術を用いて並列に処理しているので、Single Core の CPU より処理時間の短縮が期待される。このコンピュータに、メインの DAQ プログラムをインストールし、DAQ コンピュータとして使用する。

表 3.2 で BESS-Polar と BESS-PolarII で使用した CPU を比較する [7][8]。

今回の観測時期は BESS-PolarI 実験の時よりもさらに太陽活動極小期にあたるため、イベントレートが高くなることが予測される。そのため、DAQ の処理速度の向上が必要不可欠である。さらに USB 機器を多量に使用するので、USB 拡張ボード（USB ポートが 4 つ）を 2 枚使用し、CPU ボードの 2 ポートと合わせて USB ポートは合計 10 ポート (=4×2+2) 搭載している。

今回は 20 日間の長期間の観測を想定し、さらに、太陽活動極小期であるために Trigger Rate も上昇すると予測されていた。1 Event のデータサイズが約 3 kB、Trigger Rate の予測が 3 kHz、観測日数が 20 日間と予測されていたため、HDD は約 15 TB は必要であったので、HITACHI 製の SATA 接続の 1TB のディスクを外付け HDD として 16 台使用し、16TB の容量を確保した（図 3.6）。DAQ が収集したイベントデータ、およびモニターデータは全てこの HDD に保存される。複数の HDD に同時に書き込むことはないので、使用する 1 台にのみ電源を入れて、その他の電源は落としておくことにより、電力消費を抑えている。HDD の電源の ON/OFF も DAQ が管理しており、使用している HDD の容量が設定値以下になると自動的に、マウントを解除して電源を切り、次の HDD に切り替えるようになっている。PolarI ではこの機構にバグが残っていたため、残量を監視しつつ手動で切り替えていたが、デバッグに成功したので、今回は非常に安定したほぼ全自動の DAQ システムが完成した。

外付け HDD はデータ保存用なので、OS 及び DAQ プログラムは Compact Flash にインストールする。PolarI では、256MB の Compact Flash を使用していたために、容量内に納めることのできる OS として、Slackware 9（kernel 2.4）を採用していた。今回は、4GB の Compact Flash にアップグレードしたので、容量制限はなく

なったので、サイエンスの分野で定評のある Scientific Linux 5（kernel 2.6）を採用した。

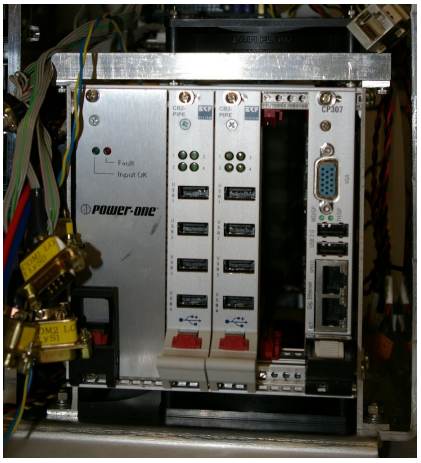

図 3.5: DAQ コンピュータ cPCI

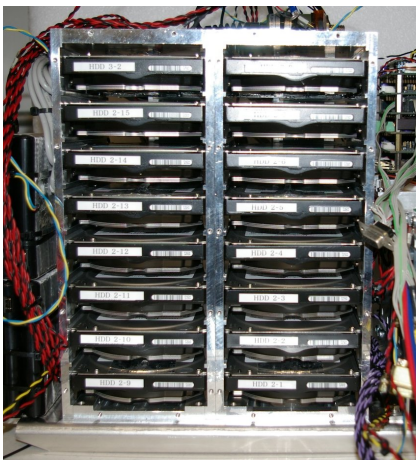

図 3.6: Data Storage (16 TB)

3.2.2 PC104

PC104 は図 3.7 に示す様な 10cm 四方の立方体の PC で、電源コントロール、地上との通信、モニターデータの収集に使用する。OS は Slackware 9 を採用し、256MB の Compact Flash 内にインストールされている。HDD は持っていないので、モニターデータ等は保存せず、cPCI および地上に送信する（図 3.8）。

電源管理は LON を通して行われる。PC104 および各電源モジュールには LON チップが搭載されており、規程のコマンドを送信することにより、電源の ON/OFF を行う。

PC104 が集めたモニターデータや、LAN を介して cPCI から送られてくるモニターデータ（と一部のイベントデータ）は、NFS の技術により、衛星等を使用して地上に送信される。

cPCI、PC104 ともに立方体に近い形をしているが、これらコンピューターはまわりが真空・低温になるのを防ぐと同時に、磁場がかけられている検出器の近くに置かれるので、磁場の影響をなくすために Data Vessel と呼ばれる釜の中に格納する必要がある。そのために、スペースを節約するために、この様な形のものが採用された。図 3.9 に、cPCI、PC104、HDD を組み上げた様子を示す。Data Vessel の中は常温・常圧に保たれているので、コンピューターが低温・低圧下にさらされて停止する心配はない。

3.2.3 Front End Modules

- FADC(=Flash ADC)

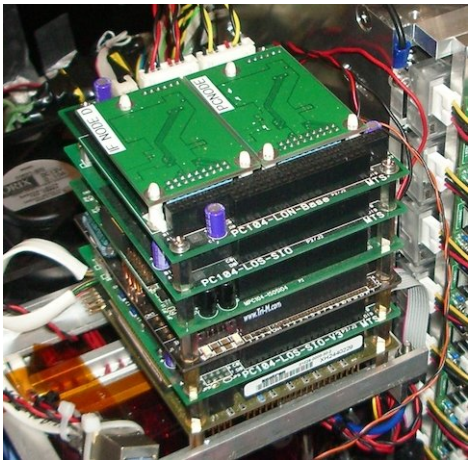

図 3.7: 通信・電源コントロール用コンピュータ　PC104

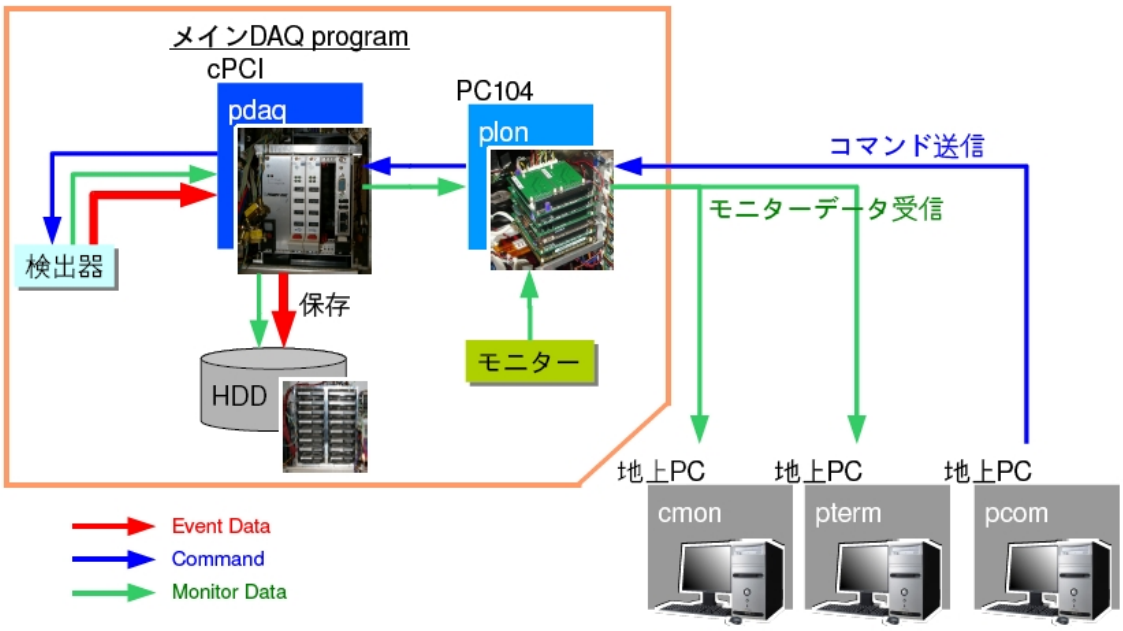

図 3.8: PC104 は地上と DAQ との中継役を担う

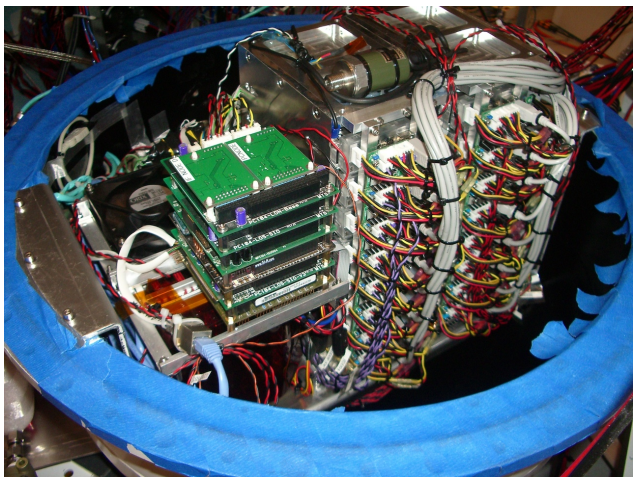

図 3.9: cPCI, PC104, HDD を組み上げ、Data Vessel に格納した様子

FADC は JET/IDC Chamber の信号を読み出すのに用いる。全部で 24 台使用し、すべて USB で接続する (図 3.10)。イベントデータの大部分を占めるのが、この FADC からのデータである。

FADC には「Raw モード」「Compress モード」の 2 つのサンプリングモードがある。Raw モードでは、チェンバーからの波形情報をすべて保存するため、データサイズが非常に大きくなる。Compress モードでは、波形情報を圧縮して転送するので非常に軽量で、1 イベントのデータサイズは固定されている。フライトではこの Compress モードを用いてサンプリングする。

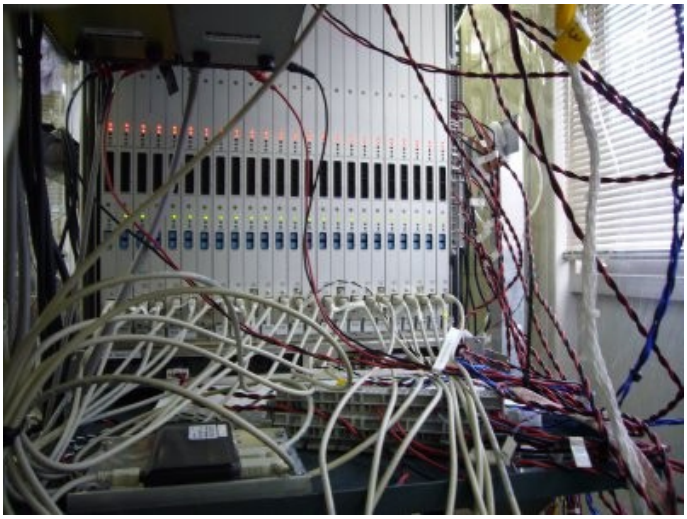

図 3.10: 24 枚の FADC ボード

- TDC

TDC(=) はTOF PMTからの信号をデジタル化しタイミング情報としてTrigger ボードに送信する。

- QDC

  QDC(= Charge(Q) Digital Converter) はTOF ／ ACCの電荷を積算しデジタル化する。

- Trigger Board

  Trigger ボードは、TDCから送られてきた信号を元にトリガーを発生させる。UTOF+LTOF、UTOF+MTOFといったトリガー条件はこのボード内部で設定されている。

- Clock Generator

  Clock Generatorは各エレクトロニクスにクロックを提供する。全てのエレクトロニクスはClock Generatorで生成されるクロックを基に動作する。

第4章　BESS-PolarIIでのDAQの改良

今回の観測は前回よりも長期間行うことが計画されており、さらに、太陽活動極小期にあたるため、イベントレートが増えることが予想される。BESS-PolarI では、DAQ の Dead Time は 20% ぐらいであり、今のシステムのままでは Dead Time が増え、期待している統計量を確保できない可能性がある。より、高速で安定した DAQ プログラムが求められた。そのため、BESS-PolarII では以下の改良を施した。

1. FADC 接続の改良
2. USB デバイスドライバーの改良
3. 通信に関する改良
   (a) IRIDIUM 用パケットの作成
   (b) コンソール版モニタープログラムの開発
   (c) オペレーションコマンドの追加
4. 自動化に関する改良
   (a) パケットスタックサイズの増量
   (b) HDD 切り替え処理のバグ修正

この章では、これらの改良について説明する。

4.1　FADC 接続の改良

BESS では、FADC は全部で 24 台使用している。しかし、Linux の USB デバイスドライバーは同時に 16 台までしか認識できない。そのため、BESS-Polar 実験では 3 台を 1 セットとし、1 台をマスター、2 台をスレーブとしてスレーブとマスターは Blackplane で接続し、マスターのみ cPCI と USB で接続していた。スレーブのデータは Backplane を通してマスターにされ、マスターが 3 台分のデータをまとめて USB で cPCI に転送する。

これの方法を採用することにより、cPCI に USB で接続する FADC の数は 8 台ですむため、Linux のデバイスドライバーの認識問題は回避されていた。（図 4.1(a)）

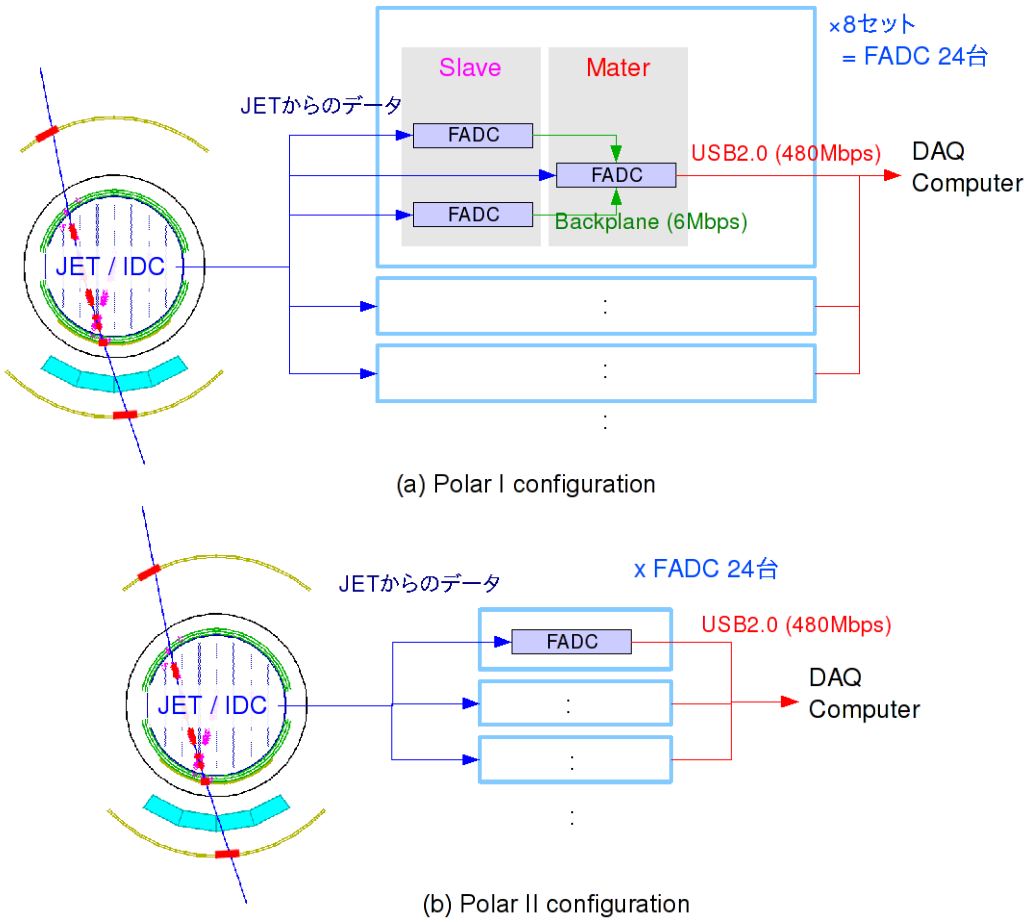

図 4.1: FADC 接続の概念図。BESS-PolarI の時の設定 (a) と BESS-PolarII で変更した設定 (b)

さらに、USB 機器は他の機器に比べて電力の消費量が多いので、少電力化の効果もあった。

しかし、この方法の欠点は Blackplane の転送速度が遅いことである。表 4.1 に転送速度の比較を示す。

表 4.1: Backplane と USB2.0 の転送速度の比較

	Backplane	USB2.0
転送速度	8 Mbps	460 Mbps

BESS-PolarI 実験では、Dead Time が 20% くらいであったが、原因の一つに Blackplane の転送速度の遅さにあると予測される。さらに、今回は太陽活動極小期にあたるので、イベントレートが増えることが予想される。そのため、前回のシステムのままでは、Dead Time が増えることが容易に想像でき、期待している統計量を確保できない可能性がある。

そこで、BESS-PolarII では FADC 接続の見直しをこない、マススレーブ接続をやめて、全ての FADC を USB 接続に変更した。これにより、Backplane の転送速度のネックはなくなり、高速なデータ転送が実現できる。

FADC は全部で 24 台あるが、cPCI の FADC に使える USB ポートは 4 ポートしかないので、USB-HUB を多段に接続して使用することにした。図 4.2 に FADC の USB 配線図を示す。

USB-HUB には、System TALKS 社製の SUGOI HUB を使用した [9]。これを採用した理由は、ハブを多段接続し、大量の USB 機器を動作させる必要があるために、出力電流が大きいものを使用する必要があった。さらに、外部電源も使用することができる。

また、HUB は Pressure Vessel 内、磁場中に置かれるので、磁場中でも安定に動作すると評判があったのも、理由のひとつである。

4.2 USB デバイスドライバーの改良

4.2.1 kernel 2.6 への対応

BESS-PolarII では DAQ コンピューターの cPCI の OS として、Scientific Linux 5(以下、SLC5) を採用した。BESS-PolarI では、Slackware 9 が採用されたが、当時は、OS をインストールする Compact Flash の容量が 256MB しかなかったために、これに収まる Linux として Slackware が採用された。しかし、今回は 4GB の Compact Flash を使用したので、容量の制限はなくなり、SCL5 を採用した。また、HDD が IDE から SATA に変更され、kernel2.4 では SATA が認識できない可能性があったので、kernel2.6 の OS が選択された。

Slackware 9 の kernel は 2.4 なので、USB ドライバーも 2.4 用に作られている。しかし、SLC5 は kernel 2.6 なので、ドライバーを 2.6 用に作り直した。kernel 2.6 に

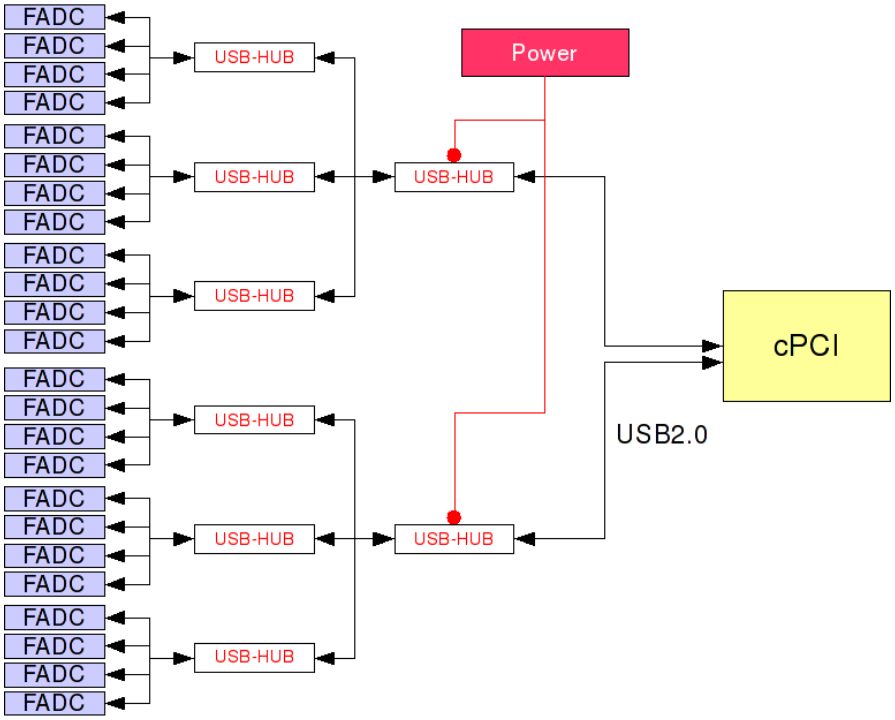

図 4.2: 全ての FADC を USB 接続にした配線図。cPCI 本体の FADC 接続に使用できる USB ポートは 4 ポートなので HUB を多段に接続して 24 台すべてを接続する。

なり、URB 構造体と一部の関数の定義が変更になったので、それに合わせてソースコードを改良した。

4.2.2 ドライバーの複製

1 つの USB ドライバーが同時に認識できる機器の数は 16 台だが、これを変更するにはカーネルそのものを修正し、再構築しなければならない。その作業は非常に大変で、修正に失敗すれば OS そのものを破壊しかねない。そこで、USB ドライバーを複製し、2 つのドライバーで分担して認識させることで、17 台以上の機器を認識させることにした。

USB が PC に接続されると、udev という hotplug 機構が動作し、USB 機器に書き込まれている Vender ID や Product ID など様々な情報・条件を元に、適切なドライバーをシステムにロードする。BESS の FADC 用の USB ドライバーは Vender ID と Product ID を参照してドライバーがロードされる。2 つのドライバーを別々に動作させるには、それぞれが認識する Vender ID, Product ID を別々にしなくてはならない。オリジナルと変更した Vender ID, Product ID を表 4.2 に示す。

また、FADC にも同じ ID を記録しておく必要あるので、8 台をオリジナルのドライバー、16 台を複製後のドライバーで認識させることとし、FADC の基板上の ROM に変更した Vender ID, Product ID 書き込み、別々に認識させ、24 台全てを USB 接続で認識させることに成功した。

表 4.2: FADC の Vender ID, Product ID

	Vender ID	Product ID	台数
オリジナル	0x0817	0x0204	8 台
複製	0x0818	0x0205	16 台

なお、FADC の分け方を 12+12 台ではなく、8+16 台にしたのは、他にも USB 機器を使用するからである。USB 機器は FADC の他にも、MU2、Clock Generator を使用するので、オリジナルのドライバーは、FADC 8 台、MU2 6 台、Clock Geneartor 1 台の合計 15 台、複製したドライバーは、FADC 16 台、という割り振りで使用した。

4.3 通信に関する改良

4.3.1 IRIDIUM パケットの作成

気球との通信に使う衛星、TDRSS と IRIDIUM とでは、TDRSS の方が通信速度が速い。IRIDUM は 15 分に一度 256 Byte 送信するのみで、それ以外のパケットは破棄される。そのため、通常フライトオペレーションには TDRSS 衛星の通信網を用いられる。しかし、TDRSS がブラックアウトしている間は TDRSS を使用できないので、IRIDIUM を使わざるおえない。

BESS-PolarI でも、IRIDIUM は使用していたが、重要度の低いデータばかり送られてきて、DAQ が正常に動作し続けているのかを判断するのが困難だった。よって、BESS-PolarII では、DAQ および検出器が正常に動作しているか判断できる最小限のモニターデータのみを 256 Byte 以内のパケットにまとめ、IRIDIUM 用のパケットとして送信することにした。これにより、常に必要な情報のみ送信しているので、IRIDIUM 通信に切り替わっても、DAQ が動作していることが確認できる。

IRIDIUM パケットに詰めたモニターデータを以下に示す。

- DAQ 系 - データサイズ、イベントレート、CPU・メモリ使用率など
- JET/IDC 系 - ワイヤー電圧・電流、GAS など

4.3.2 コンソール版モニタープログラムの開発

気球から送られてくるモニターデータを監視するプログラムとして ROOT+GUI で構築された「pmon」がある。しかし、このプログラムは長時間使用し続けると動作が非常に遅くなり、モニタリングに支障をきたすほどであった。そこで、より軽量なモニタリングプログラムが必要となった。

また、モニターデータは衛星から Palestine の Whitesuns を介してインターネットで南極まで送られてくるのだが、南極のネットワーク回線は非常に弱いので、モニ

ターデータの送信が停止することもある。そのため、外部の PC に ssh でリモートログインした状態で使えるものがあれば便利である。

これらの要求から、Linux 端末内で動作するコンソール版モニタープログラム「cmon」を開発した。cmon の動作イメージを図 4.3 に示す。モニター画面は端末操作用ライブラリ ncurses を用いて構築した。

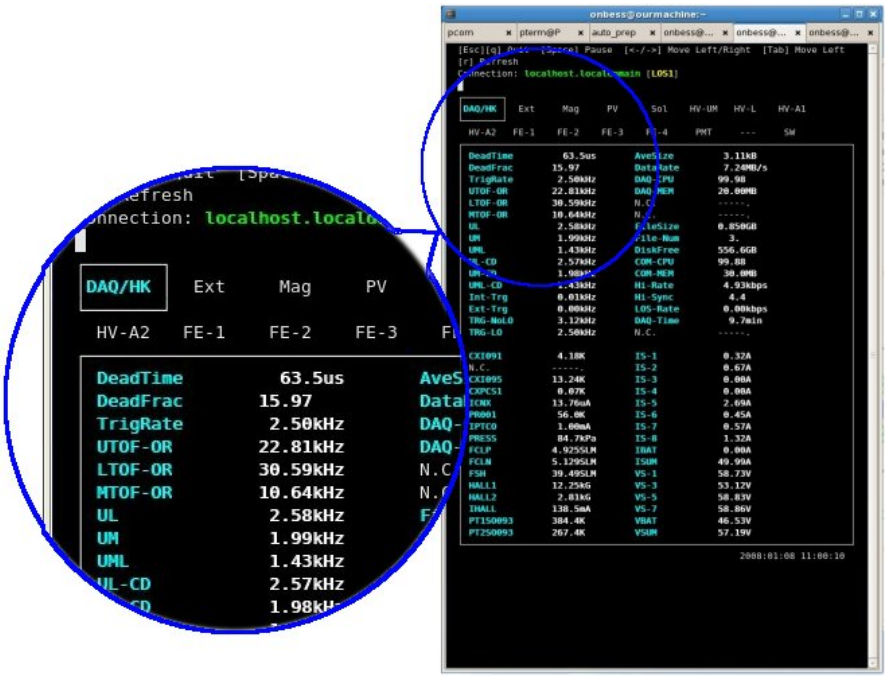

図 4.3: コンソール版モニタープログラム「cmon」のスクリーンショット

cmon は GUI を必要としないので、端末に内で軽快にストレスなく動作し、X Window が起動していない Linux マシンでも使用することができる。

4.3.3 オペレーションコマンドの追加

BESS-PolarI の反省から、より安全で安定したフライトを実現するためにオペレーションコマンドを追加した。コマンドを簡略化したことにより、フライトオペレーションによる Dead Time を減らすことができた。表 4.3 に追加したコマンドをまとめる。

1. HV マスクコマンド
   PMT が故障し使用できなくなったとき、それらの PMT には HV をかけないのだが、BESS-PolarI の時は、必要なチャンネルを個別に指定して HV をかけていた。しかし、その作業には多量のコマンドを送信しなければならないため、非常に時間がかかる。そこで、あらかじめ特定のチャンネルを無効にして、それ以外のチャンネルに 1 コマンドで HV をかけられる様なコマンドを作成した。

表 4.3: 新たに追加したオペレーションコマンド一覧

コマンドタイプ	コマンド名	説明
HV マスク	@hkil	指定したチャンネルの HV を無効にする
	@hrev	指定したチャンネルの HV を有効にする
	@hsta	HV の有効/無効の状態を表示する
HV コントロール	@hdon	Drift Chamber Wire の HV を ON にする
	@hdoff	Drift Chamber Wire の HV を OFF にする
	@hson	Sense Wire の HV を ON にする
	@hsoff	Sense Wire の HV を OFF にする
cPCI 電源コントロール	@cpon	cPCI の電源を ON にする
	@cpof	cPCI の電源を OFF にする
Clock Generator	@rset	Clock Generator をリセットする
	@ctst	テストコマンド。エコーを返すだけ
	@trig	トリガーの ON/OFF を切り替える
	@disb	特定のチャンネルを有効/無効にする

2. HV コントロールコマンド
   HV コントロールは GUI のプログラムで行えるが、より素早いオペレーションが行えるように、コマンドとしても実装した。

3. cPCI 電源コントロールコマンド
   cPCI そのものがコマンドを受付なくなった場合に備えて、外部から電源をコントロールし cPCI を回復させられるようにした。

4. Clock Generator コマンド
   Clock Generator も今回から UBS 接続になったので、DAQ から直接コントロールできるようになった。

4.4 自動化に関する改良

4.4.1 パケットスタックサイズの増量

CSBF で行った噛み合わせ試験の段階で、DAQ が非常に短時間で止まるという問題が発生した。デバッグの結果、原因はスタックサイズが小さすぎたためだったので、容量を増やした。

各検出器からのパケットは非同期的に USB を通じて cPCI に送られる。そのため、全ての機器からのパケットが揃いイベントビルドを行うまでは、メモリに保持しておかなければならない。DAQ では、それらは一時的にスタックに溜められる。このスタックの容量が足りなかったために、パケットが揃いきる前にスタックがいっぱ

いになり、それ以降のパケットを受け入れられなくなったために、DAQ が止まっていた。

4.4.2 HDD 切り替え処理の自動化

BESS の DAQ では 1TB の HDD を 16 台積み、使用する 1 台のみ電源を入れる。そのため、使用中の HDD の容量がなくなったら、次のディスクに切り替える必要がある。 BESS-PolarI の時に、HDD の残量がある設定値以下になったら、次の HDD に切り替える機構が組み込まれた。しかし、HDD のデバイスから読み出すサイズと、プログラム内で設定しているサイズの単位が食い違っていたために、この機構が機能せず、手動で切り替えるしかなかった。今回のフライトではこのバグを修正したので、ほぼ全自動の DAQ が出来上がった。

4.5 DAQ 改良のまとめ

今回の観測時期は前回より太陽活動極小期にあたるので、Trigger Rate の上昇が予測され、それに耐えうる様に DAQ を改良してきた。さらに、CPU も Dual Core のものに変更したので、性能の向上が向上した。

BESS-PolarI と BESS-PolarII の DAQ の性能の比較を表 4.4 に示す。

表 4.4: DAQ の性能の比較

	BESS-PolarI（2004 年）	BESS-PolarII（2007 年）
Trigger Rate	1.4 kHz	3.4 kHz
Dead Time	20%	23%
Data Size	2.1 TB	13.5 TB
Event Size	2.3 kB	3.0 kB
Event Number	900 Million	4700 Million
Observation Time	8.5 days	24.5 days
イベント処理能力	2.6 MB/s	7.8 MB/s

Trigger Rate は 1.4 kHz から 3.4 kHz と 2 倍以上に上昇している。やはり、太陽活動極小期の影響が出ているものと思われる。

Trigger Rate が大幅に上昇しているにもかかわらず、Dead Time はわずかに 3%増えただけである。イベントサイズも 2.3 kB から 3.0 kB と増えているので、もし、FADC や USB の改良、CPU の刷新を行わなければ、Trigger Rate が増えた分だけ Dead Time が増え、Dead Time は倍以上になっていたと予測される。

よって、これらのことより DAQ の性能は向上されたと考えられるので、単位時間にどれくらいのサイズのデータを処理できたのかを以下の式で評価する。

$$\begin{aligned} \text{イベント処理能力} &= \text{TriggerRate[Hz]} \times \text{EventSize[Byte]} \\ &\quad \times (100 - \text{DeadTime[\%]}) \times 0.01 \end{aligned} \tag{4.1}$$

この式で DAQ のイベント処理能力を計算すると、BESS-PolarI は 2.6 MB/s であるのに対し、BESS-PolarII は 7.8 MB/s と約 3 倍も性能が向上させることに成功した。また、各種自動化機構の改良・修正により、BESS-PolarII 実験では非常に安定した高速な DAQ を実現することができた。

第5章　GEANT4による測定器シミュレーション

BESS実験では反陽子のFluxを求めるのが目的としており、観測データに検出器の検出効率や大気補正など様々な補正ををかけ合せてFluxを求める。BESSのシミュレーションではその内の検出効率 (Efficiency) を求める。

シミュレーターはBESS-PolarIまでのものはGEANT3で開発されていたが、BESS-PolarIIのものは存在しないかったので、本研究でGEANT4(= GEometry ANd Tracking 4) を用いて開発した。

5.1　BESSでの反陽子解析

5.1.1　BESS-PolarIIのためのシミュレーション

BESSのシミュレーションプログラムは、前回のBESS-PolarIのものまではすでにGEANT3(FORTRAN) を用いて開発されている。しかし、BESS-PolarII測定器のシミュレーションはまだ存在しないので、GEANT4(C++) を用いて開発することにした。

5.1.2　GEANT4(C++)による開発

今までのBESSのシミュレーションはGEANT3を用いて行われてきた。しかし、GEANT3のメンテナンスはすでに終了している。また、GEANT3はFORTRANで構築されているが、今日ではFORTRAN言語の教育はほとんど行われなくなってきている。そのため、この先FORTRANの技術者は減っていくと思われるので、GEANT3で構築されたプログラムはメンテナンスが困難になっていく。

そのため、現在の高エネルギー業界のシミュレーションは主にGEANT4(C++) を用いて構築されている。そこで、BESSでも今回のBESS-PolarIIのシミュレーションはGEANT4を用いて構築することにする。

5.1.3　シミュレーションの位置づけ

BESSでは、主に反陽子のFluxを求めることが目的である。解析した観測データに検出効率、大気補正をかけ合わせてFluxを算出する。シミュレーションでは検出効率、すなわち検出器に降り注いだ全粒子の内何割を捕まえられたのかを見積もる。

以下の式は、観測データとシミュレーション等で求めたデータを使って Flux を求めるまでを表している。観測データからは、$N_{obs}, N_{BG}, T_{live}, \epsilon_{PID}, \epsilon_{acc}$ シミュレーションでは、$S\Omega, \epsilon_{single}$ を求める。その他のパラメーターはあらかじめ別に求めておく。

$$\mathrm{Flux} \cdot \mathrm{dE} = (\mathrm{N_{obs}} - \mathrm{N_{BG}}) \cdot \frac{1}{\epsilon} \cdot \frac{1}{\mathrm{S\Omega} \cdot \mathrm{T_{live}}} \cdot \frac{1}{\eta + \mathrm{R_{air}}} \tag{5.1}$$

$$\epsilon = \epsilon_{rec} \cdot \epsilon_{single} \cdot \epsilon_{PID} \cdot (1 - \epsilon_{acc}) \tag{5.2}$$

N_{obs}	:	観測データ	ϵ_{rec}	:	Reconstruction Efficiency
N_{BG}	:	Background	ϵ_{single}	:	Single Track Efficiency
$S\Omega$	:	Geometrical Acceptance	ϵ_{PID}	:	Particle Identification
T_{live}	:	Total Live Time	ϵ_{acc}	:	Accidental Event の確率
$\eta + R_{air}$	:	大気補正			

N_{obs} は解析で求めた反陽子（陽子）の数、N_{BG} はその Background である。T_{live} は観測した時間なので、フライト時間に DAQ の Dead Time などでロスした時間を差し引いて求める。$S\Omega$ は Geometrical Acceptance、ϵ_{single} は Single Track Efficiency ϵ_{single} である。$\eta + R_{air}$ が大気補正で、η は降り注いだ粒子が残留大気により消滅せず生き残る確率、R_{air} は 2 次粒子が生成されて交じった割合である。地上より大気の影響は少ないとはいえ、7 g/cm^2 くらいは残っているので、大気による増減も考慮しなければならない。ϵ_{rec} はトラックとなるはずのものをトラックとして再構成できる割合、ϵ_{PID} は粒子を正しく特定できる割合、ϵ_{acc} は本来はイベントしてはなり得ないのに、同タイミングで他の粒子が入り、イベントとして捉えてしまう確率である。

5.1.4 Multi Track の解析

BESS では低エネルギーの反陽子を探索しているため、その様な粒子は測定器下部まで到達せずに途中で崩壊するものが多い。しかし、崩壊によって発生した二次粒子がトラックを構成しイベントとして認識できる場合がある。この様なイベントをマルチトラックイベントと呼ぶ。現在の Flux の解析では、マルチトラックイベントは使用していない。しかし、JET/IDC チェンバーを通過した後に崩壊した様なイベントの場合はエネルギーを再構成できるので、この様なイベントを使用することで低エネルギー領域の統計を稼ぐことができるので、一次起源反陽子に対してより高い制度の解析ができる可能性がある。この解析手法はまだ確立していないので、シミュレーションを改良し、マルチトラック解析を行えるようにする必要がある。

5.2 GEANT4について

5.2.1 GEANT4とは何か

GEANT4とは粒子と物質の相互作用をシミュレーションする汎用ソフトウェア・ツールキットである。高エネルギー物理学実験で使われる測定器の振る舞いをシミュレーションするのを第一義として開発されたが、設計段階から他の分野での応用も考慮されていた。現在では宇宙、医療をはじめとする広い分野でも使用されている。

5.2.2 GEANT4によるシミュレーション

GEANT4はシミュレーションしたい事象に含まれている粒子を、物質および外部電磁場との相互作用を考慮しつつ、次のいずれかの条件が成り立つまでTransport（輸送）する。

1. 運動エネルギーがゼロになるまで
2. 相互作用により消滅するまで
3. ユーザか指定するシミュレーション空間の境界（世界の果て）に到達するまで

さらに、シミュレーションのさまざまな段階で、ユーザがデータを処理できる様に構成されている。

- Sensitive Detectorとして登録した測定器に粒子が入った際に、その時のHit情報を処理する
- 粒子をTransportする各ステップでステップ情報を処理する
- １イベントの最初と最後にデータ処理する

等である。またGEANT4は以下のことをサポートしている。

1. シミュレーションの対話的処理、および、バッチ処理
2. シミュレーション過程を可視化する各種グラフィック・ツール
3. シミュレーションのチェック、デバッグ・ツール

5.2.3 ユーザーによるプログラミング

GEANT4を用いてシミュレーションをするには、ユーザーはまず以下のことをプログラムしなければならない。

- 測定器の構造情報
- シミュレーションしたい事象に含まれる粒子の種類、始点と運動量ベクトル

- 考慮する粒子および相互作用の種類

さらに、以下のことも必要に応じて設定しなければならない。

- 外部電磁場（および、分布情報）
- シミュレーションの各段階でユーザが行いたいデータ処理

GEANT4 はシミュレーション用 API なので、これらのことを考慮してプログラミングした上で、各実験にあるようにパラメーターなどを調整する必要がある。

5.3 BESS-PolarII 測定器シミュレーション

5.3.1 シミュレーションの流れ

シミュレーションはおおまかに、打ち込む宇宙線を生成する「Event Genaration」、生成した宇宙線を打ち込み検出器をシミュレートする「Simulation」デジタル化および検出効率を含める「Digitization」に分類できる。シミュレーションの流れを図 5.1 に示す。BESS のシミュレーターでは Event Generation、Simulation を行う。

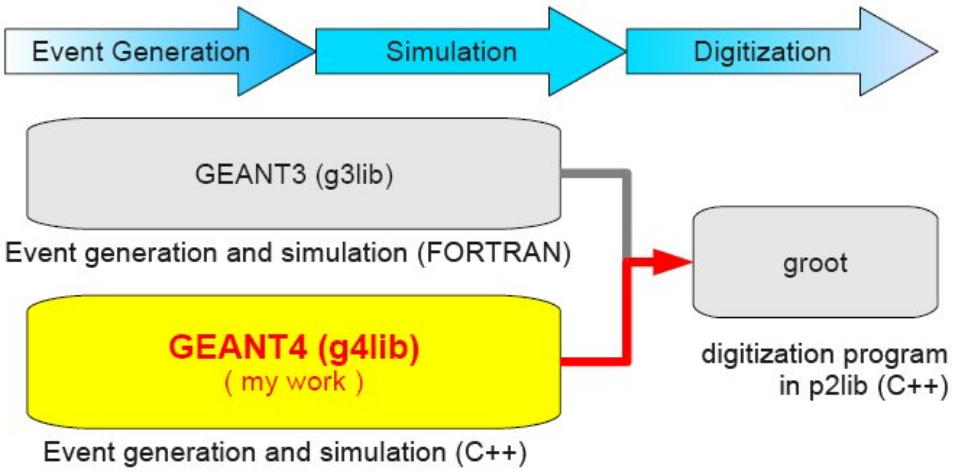

図 5.1: シミュレーションの流れ

1. Event Generate

   Event Generate ではシミュレーションで検出器に打ち込む宇宙線を生成する。GEANT4 の全物理プロセスを無効にして、全方位から様々なエネルギーの宇宙線を打ち込んでみて、イベントとなり得そうな「UTOF およびマグネット内を通過したイベント」のみを選択・保存する。実際にシミュレーションのトリガー条件とは異なるが、2 次粒子が他の TOF カウンターに Hit としてイベントなる可能性もあるので、通常のトリガー条件より甘くしている。

2. Simulation

   Simulation では、Event Generation で生成したイベントを読み込み順にシミュレートする。Simulation では「ALL-OFF」「ALL-ON」の 2 つのモードを実行する。

   (a) ALL-OFF
   全ての物理プロセスを無効にしたモード。Event Generation と同じ状態。トリガーが発生するかどうかは、検出器の geometry のみで決まるので、Geometrical Acceptance を求めることができる。

   (b) ALL-ON
   全ての物理プロセスを有効にしたモード。通常のシミュレーションモード。Single Track Efficiency などを求めることができる。

GEANT4 のシミュレーションプログラムでは、シミュレーション結果がバイナリデータで出力される。出力されたデータを、Digitization Program で検出器応答などの効果を入れる。
詳しくは、6.2, 6.3 で説明する。

5.3.2 シミュレーションで求めるもの

BESS のシミュレーションでは式 (5.1)(5.2) のうち

- Geometrical Acceptance ($S\Omega$)
- Single Track Efficiency (ϵ_{single})

の 2 つを求める。
Geometrical Acceptance は、検出器がで捕まえることのできる宇宙線の量を表す値である [10]。詳しい求め方は 7.2 で説明する。この値は、測定器の幾何学的構造のみで決定される。算出する際には粒子が他の物質と相互作用しないように ALL-OFF モードを用いて求める。
Single Track Efficiency は、イベントとして記録されたものの内、single track として認識できる割合である。測定器を通過中に反応を起こして、track として認識できない場合もあるので、その割合を見積もらなければならない。

5.3.3 シミュレーションプログラム

本研究では BESS-PolarII 測定器の GEANT4 API を用いたシミュレーターとして「g4lib」を開発した。また、Digitization Program の「groot」および、イベントディスプレイ「gevt」も g4lib の開発に合わせて、改良・開発をしたので合わせて第 6 章で説明する。

第6章　g4libライブラリの開発

g4lib ライブラリは、本研究で開発したシミュレーションのメインライブラリであり、本研究のメインテーマである。

この章では、開発した g4lib ライブラリについて、検出器の設定、アルゴリズム、Cross Section の改良について詳しく説明する。また、Digitization Program やイベントディスプレイも改良したので、それらについても説明する。

開発には、GEANT4.9.1 patch2, ROOT 5.18.00 を使用した。

6.1　Detectors

BESS-Polar 実験の検出器は、UTOF, MTOF, LTOF, JET, IDC, ACC の 6 つを使用する。これらを、Sensitive Detector として GEANT4 に登録し、シミュレーションデータを取得する。GEANT4 で構築した検出器の geometry を図 6.1 に示す。上記の検出器の他にも、超伝導マグネット、Cryostat, Endcap, TOF のサポートなど様々な物体が実装されている。実際のフライト中では、検出器の周りには少量の空気（7g/cm^2）が存在するが、大気補正の効果は別に計算して補正するので、ここでは真空で満たしてある。

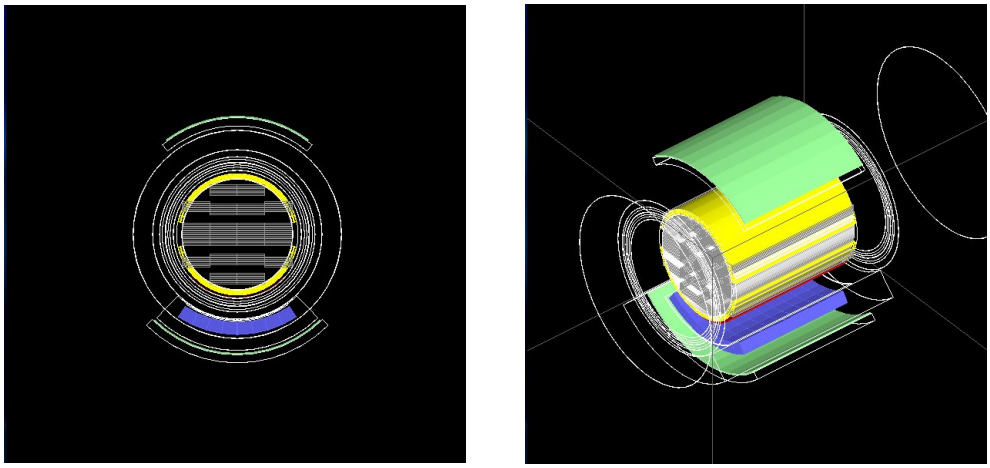

図 6.1: GEANT4 でシミュレートした BESS-PolarII 測定器

6.1.1 TOF (UTOF, MTOF, LTOF)

TOF 用の Scintilator にはプラスチック・シンチレーターを使用しているので、物質には密度 1.032 g/cm^3 の CH を用いた。また、TOF は円筒状の検出器の上下に円形に配置するので、各 TOF Paddle の形は正確な直方体ではなく、内側の幅が少し狭い、四角錐型をしている。

TOF(UTOF, LTOF, MTOF) から取得するデータフォーマットを表 6.1 に示す。Copy Number は TOF の各 Paddle を区別するために使用する。UTOF は全部で 10 本、LTOF は 12 本、MTOF48 本あるので、Copy Number でそれぞれを区別する。Track ID はその Hit が所属する Track の ID、Hit Position は Hit した位置、Energy Deposit は粒子が TOF 内で落としたエネルギー、Step Length は TOF 内を通過した距離である。

表 6.1: TOF から取得するデータ

データ	サイズ [Byte]	説明
Copy Number	4	Detector の Copy Number
Track ID	4	その Hit が所属する Track の ID
Hit Position	4×3	Hit 位置 (x,y,z)
Energy Deposit	4	TOF 内で落としたエネルギー
Step Length	4	粒子が TOF 内を走った距離
Hit Time	4	Hit 時間

Track は複数の step から構成されるので、TOF 内部でも複数 Hit に分かれる可能性はある。しかし、実際の信号が 1 つの TOF から複数でることはありえないので、そのような Hit は 1 つに結合して保存する。また、実際は PMT でシンチレーターの信号を取得するが、シンチレーション光を正確に再現するとシミュレーションに非常に時間がかかる. 実験データの解析では、データに対してキャリブレーションを行って各種分解能を求めた上で真の物理量を見積もる。シミュレーションでは逆に初めに物理量がわかっているので、PMT は再現せず物理量から分解能や検出器応答の効果を混ぜて実験データをシミュレートする。

6.1.2 JET/IDC

JET/IDC はワイヤーを平行に張り巡らせて、ワイヤーに集まった信号をデータとして取得しているのだが、シミュレーションでは、各ワイヤーの位置を中心にした領域を仮想的に Box として定義して、その内部での座標やエネルギー損失を取得する。Box 内部は密度 0.0017793 g/cm^3 の CO_2 で満たされており、JET/IDC 共通である。

JET/IDC から取得するデータフォーマットを表 6.2 に示す。Copy Number は各ワイヤーに一意に割り振られた番号。Track ID はその Hit が所属する Track の ID. Hit Position は本来 Track の中でワイヤーに最も近い位置を定義すべきだが、Box と

して定義してしまうと得られる位置は、Track が Box に入った位置と Box から出た位置のみである。また、Box 自体あまり厚くないものなので、入出位置の中点を Hit 位置として保存する。Energy Deposit は Box 内でのエネルギー損失、Wire Number は各ワイヤーに割り当てられた ID である。Copy Number はシミュレーション内で独立に決められたものだが、Wire Number は実際の測定器でも定義されている ID なので、実験データと比較することができる。

表 6.2: JET/IDC のデータフォーマット

データ	サイズ [Byte]	説明
Copy Number	4	各ワイヤーの Copy Number
Track ID	4	その Hit が所属する Track の ID
Hit Position	4×3	Hit 位置 (x,y,z)
Energy Deposit	4	Box 内部でのエネルギー損失
Wire Number	4	ワイヤーナンバー

6.1.3 ACC

ACC (= Aerogel Cherencov Counter) は、各 Box を Aerogel として定義し、マグネットの下部、LTOF の上部に 3×4 = 12 blocks 配置する。物質には密度 0.0775 g/cm^3 の SiO_2 を使用する。実際は、Aerogel から発生した Cherencov 光を PMT でとらえて信号を出力するが、BESS のシミュレーションでは Cherencov 光までシミュレートしない。TOF と同様に Energy Deposit などの値から分解能などを混ぜ込んで実験データを再現する。

ACC から得られるデータフォーマットを表 6.3 に示す。

表 6.3: ACC のデータフォーマット

データ	サイズ	説明
Copy Number	4	各 ACC ブロックの Copy Number
Track ID	4	その Hit が所属する Track の ID
Hit Position	4×3	Hit 位置 (x,y,z)
Charge	4	通過した粒子の持つ電荷
Beta	4	通過した粒子の速度
Energy Deposit	4	ACC 内でのエネルギー損失
Path Length	4	粒子が ACC 内を進んだ距離

6.2 Event Generation

6.2.1 宇宙線の生成

加速器のシミュレーションと違い、宇宙線のシミュレーションではビーム（宇宙線）を様々な方向から様々なエネルギーで打ち込まなければならない。

BESS では、粒子のエネルギーは陽子で 0.1〜100 GeV、反陽子で 0.01〜10 GeV の範囲と広範囲なので、対数に対して一様分布になるように乱数を生成する。生成する粒子のエネルギーの範囲を E_{min}〜E_{max} とすると、$\log_{10} E_{min}$〜$\log_{10} E_{max}$ の範囲で一様な乱数を生成する。生成された乱数を R_{log} とすると、最終的にビームエネルギー E は

$$E = 10^{R_{log}} \tag{6.1}$$

と求める。

ビームの入射位置と方向は、上半球から一様に入射するように生成する。まず、検出器のまわりに仮想的な球体があると仮定し、その球体の表面上の 2 点をランダムに指定する。各点が表面上で一様に生成されるように、

$$\theta = \arccos(1 - 2 \times R_0^1) \tag{6.2}$$

$$\phi = R_0^1 \times 2\pi \tag{6.3}$$

と、θ, ϕ を乱数で決定する。R_0^1 は 0〜1 の一様乱数である。生成された θ, ϕ より、

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \sin\theta\cos\phi \\ \sin\theta\sin\phi \\ \cos\theta \end{pmatrix} \tag{6.4}$$

と、座標を決定する。この方法で決めた 2 点を結び、入射方向が下向きになるように入射位置と方向を決定する（図 6.2）。これで、宇宙線のエネルギー、入射位置、入射方向を決定することができる。

6.2.2 イベントの選択

宇宙線のエネルギー、入射位置、方向を決定した段階で一応のイベント生成は完了しているが、生成したイベントのほとんどが検出器を全く通過しない。シミュレーション時間を短縮するために、生成した宇宙線を一度実際に検出器に打ち込んでみる。この際、全ての物理プロセスは無効にして、生成された宇宙線が検出器を通過するかということのみを確認する。

イベント生成の段階でのトリガー条件は、UTOF およびマグネットを通過したかである。シミュレーションの段階でのトリガー条件は、UTOF+LTOF もしくは UTOF+MTOF であり、UTOF ＋マグネットというトリガー条件は意味のないように見える。しかし、イベント生成の段階では物理プロセスは無効にしてあるので粒子はただ磁場にしたがって曲がるだけだが、実際のシミュレーションでは全ての物

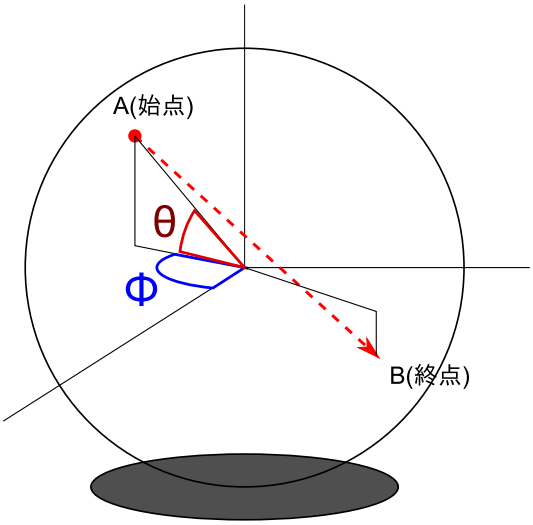

図 6.2: イベント生成の概念図。θ, ϕ を一様乱数で決定し、球体上の 2 点 A, B を決定する（半径は固定)。2 点間を向きが下向きになるように結んだとき、A 点が初期位置、矢印の向きが運動方向である。

理現象をシミュレートするので、様々な反応によって、軌跡が変化したり別の粒子が生成されたりする。それらの影響で MTOF や LTOF に Hit が生まれ、イベントとして成立する可能性もある。そのために、イベント生成の段階でのトリガー条件は本来の条件よりすこし余裕を持たせて、UTOF+マグネットにしている（図 6.3)。

6.2.3 Event Package

生成されたイベントは一旦 Event Package として HDD に保存される。Event Package には表 6.4 の情報を保存する。

表 6.4: Event Package

データ	タイプ	サイズ	説明
PDG	int	4	PDG Encoding
Event Number	int	4	イベント番号
Vertex	float	4×3	宇宙線の入射位置 (x, y, z)
Momentum	float	4×3	宇宙線の入射方向 (p_x, p_y, p_z)
Kinetic Energy	float	4	宇宙線の運動エネルギー
（合計）		36	

PDG Encoding は GEANT4 で設定されている粒子の ID である。例えば、陽子なら 2212、反陽子なら-2212 である。Event Number は Event Generation で降らせたイベントの通し番号である。最終的に Acceptance を求めるときに、何イベント降らせ

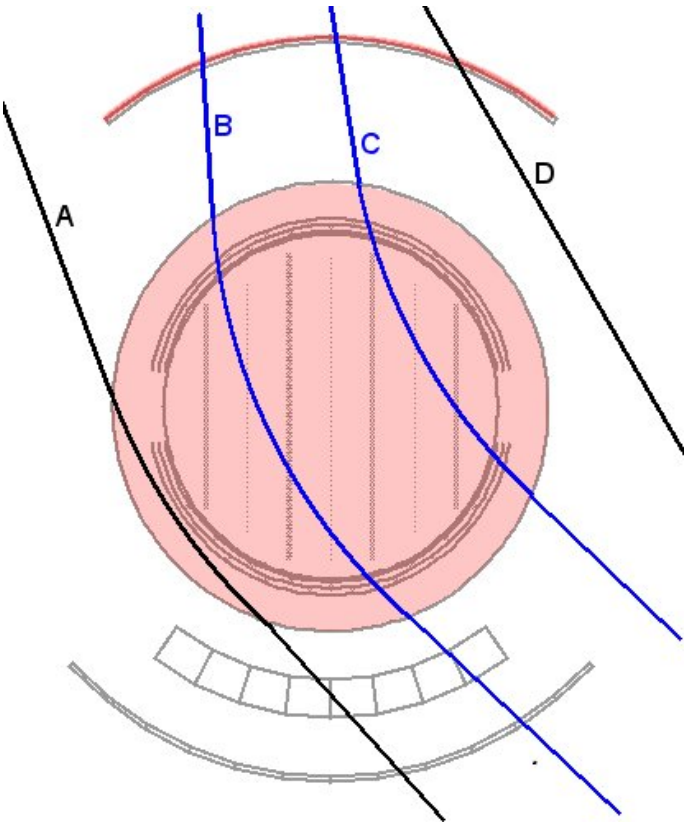

図 6.3: イベントとして選択されるためには UTOF+マグネット (赤い領域) を通過しなければならない。図のトラックのうちイベントセレクトされるのは、B, C である。どちらも UTOF+マグネットを通過している。A は UTOF を通過しておらず、D はマグネット通過していない。

て何イベント捕まえたかという情報が必要になる。その時に、このイベント番号から元々何イベント降らせたのかを逆算する。Vertex はビームの入射位置、Momentum は入射方向、Kinetic Energy は運動エネルギーである。Kinetic Energy の情報が含まれているので、Momentum は方向のみで、大きさが 1 に規格化されているベクトルである。

保存した Event Package は次に説明する Simulation の時に読み出し、ビームの設定として使用する。

6.3 Simulation

Simulation では、Event Generation で生成された Event Package を読み込んで順にシミュレートする。

GEANT4 に用意されているサンプルの物理プロセスはは加速器実験のためのものが多いので、BESS と同じように宇宙線を扱っている GLAST 実験用に用意された物理プロセスを BESS 用に調整して使用する。

シミュレーションには、ALL-OFF, ALL-ON の 2 つのモードがあり、1 回のシミュレーションでこれら全てのモードを実行する。毎回 Event Generate で生成した Event Package を読み込みシミュレーションを行う。各モードで使用する物理プロセスの一覧を表 6.5 に示す。

表 6.5: 各モードで使用する物理プロセス

Process	ALL-OFF	ALL-ON
Decay		○
conv		○
compt		○
phot		○
PhotoInelatic		○
msc		○
eIoni		○
eBrem		○
ElectroNuclear		○
annihil		○
PositronNuclear		○
muIoni		○
muBrems		○
muPairProd		○
muMinusCaptureAtRest		○
hIoni		○
HadronElastic		○
NeutronInelastic		○
HadronFission		○

HadronCapture	○
PionPlusInelastic	○
PionMinusInelastic	○
PionMinusAbsorptionAtRest	○
KaonPlusInelastic	○
KaonMinusInelastic	○
KaonZeroLInelastic	○
KaonZeroSInelastic	○
ProtonInelastic	○
AntiProtonInelastic	○
AntiProtonAnnihilationAtRest	○
AntiNeutronInelastic	○
AntiNeutronAnniihlationAtRest	○
LambdaInelastic	○
AntiLambdaInelastic	○
SigmaMinusInelastic	○
AntiSigmaMinusInelastic	○
SigmaPlusInelastic	○
AntiSigmaPlusInelastic	○
XiMinusInelastic	○
AntiXiMinusInelastic	○
XiZeroInelastic	○
AntiXiZeroInelastic	○
OmegaMinusInelastic	○
AntiOmegaMinusInelastic	○
DeuteronInelastic	○
TritonInelastic	○
ionIoni	○
He3Inelastic	○
AlphaInelastic	○
ionInelastic	○

ALL-OFF モードでは全てのプロセスを無効にする。物質と何の相互作用も起こさないので、検出器の構造のみが重要となる。このモードの結果より Geometrical Acceptance を算出することができる。

ALL-ON モードでは全てのプロセスを有効にする。完全なシミュレーションのモードである。全ての物理プロセスが有効になっているので、粒子と反粒子が同じ結果になるとは限らない。全ての効果が入っているので、このモードの結果より Single Track Efficiency を求めることができる。

イベントが有効なイベントとして HDD に保存されるためにはトリガー条件に一致しなければならない。BESS のトリガー条件は、

- UTOF + LTOF
- UTOF + MTOF

のどちらかである。U+L か U+M に Hit があればそれがどの粒子によるものかは関係なく有効なイベントとみなし保存される。

最終的に保存されるイベントは、生成したイベントの約 2.5% くらいである。

6.4 Cross Section

BESS では主に 1 GeV 以下の低エネルギー領域での反陽子を扱っているが、その領域での GEANT が持っていた Cross Section は正確ではなかったため、ビームテストの結果を用いて調整している [11][12]。各種パラメーターは結果を Fit して得た値である。g4lib も同様に陽子・反陽子の Cross Section を調整した。

6.4.1 Proton

陽子の Cross Section は表 6.6 の場合に適用される。

表 6.6: BESS Proton Cross Section の適用範囲

入射粒子	陽子
被入射物質	質量数 4 以上

陽子の Cross Section は GEANT4 のデフォルトの Cross Section に変更を加えている。Inelastic, Elastic の Cross Section $\sigma_{\mathrm{Inelstic}}, \sigma_{\mathrm{Elastic}}$[mbarn] はそれぞれ、

$$\sigma_{\mathrm{Inelastic}} = \gamma \cdot \sigma_{\mathrm{LaRC}} \tag{6.5}$$

$$\sigma_{\mathrm{Elastic}} = \gamma \cdot \sigma_{\mathrm{LaRC}} + \sigma^0_{\mathrm{Inelastic}} + \sigma^0_{\mathrm{Elastic}} \tag{6.6}$$

と、求める。$\sigma^0_{\mathrm{Inelastic}}, \sigma^0_{\mathrm{Elastic}}$ が GEANT4 から得られるデフォルトの Cross Section [mbarn] である。σ_{LaRC} は LaRC(= NASA Langley Research Center) model で求めた Cross Section である [13]。γ は correction factor で、運動量 p が 100 GeV を超える場合は 1 でなくなる。

$$\gamma = \begin{cases} C\ \log(p) + 0.5 & (p > 100\mathrm{GeV}) \\ 1 & (p \leq 100\mathrm{GeV}) \end{cases} \tag{6.7}$$

$C = 0.1085736156$ である。

表 6.7: BESS Anti-Proton Cross Section の適用範囲

入射粒子	反陽子
被入射物質	質量数 3 以上

6.4.2 Anti-proton

反陽子の Cross Section は表 6.7 の場合に適用される。

反陽子の Inelastic, Elastic の Cross Section $\sigma_{\mathrm{Inelastic}}, \sigma_{\mathrm{Elastic}}$[mbarn] はそれぞれ以下の様に求める。

$$\sigma_{\mathrm{Inelastic}} = \gamma\, \sigma'_{\mathrm{Inelastic}}\, (A/a_1)^{a_4} \tag{6.8}$$

$$\sigma_{\mathrm{Elastic}} = \gamma\, (a_2\, p^{a_5} + a_3)\, (A/a_1)^{a_6} \tag{6.9}$$

A は物質の質量数で、γ, p は陽子場合と同じである。各パラメーターは $a_1 = 26.98$, $a_2 = 205.47$, $a_3 = 210$, $a_4 = 0.63$, $a_5 = -0.81696$, $a_6 = 0.796$ である。

$\sigma'_{\mathrm{Inelastic}}$ は以下の式で求め、

$$\sigma'_{\mathrm{Inelastic}} = b_1 p^{b_2} + b_3 \tag{6.10}$$

パラメーター b_1〜b_3 は運動エネルギー K によって使い分け、0.408GeV $\leq$ K $\leq$ 15GeV の時は $b_1 = 413.983$, $b_2 = -0.159877$, $b_3 = 254.943$ であり、それ以外の場合は、$b_1 = 211.791$, $b_2 = -0.697173$, $b_3 = 454.217$ である。

6.5 Digitization

シミュレーションで作成されたデータにはまだ検出器応答の効果は含まれておらず、分解能 0 の理想的な検出器となっている。それらに Digitization Program によってキャリブレーションなどで求められた各検出器ごとの分解能などの効果を入れ観測データを再現する。

6.5.1 TOF

TOF では、電荷 Q_{TOF}[MIP](QDC)、時間 T_{TOF}[ns](TDC) を求める。

まずは電荷 Q_{TOF} を求める。

$$A = p_0 + (1 - q_0) e^{-\frac{z}{p_1}} \tag{6.11}$$

$$Q_{TOF} = A \cdot \kappa_{QDC} \cdot E_{loss} + R_{gaus} \cdot \sigma_q \tag{6.12}$$

E_{loss}[MeV] が Energy deposit と z[mm] が Hit の z 位置で、シミュレーションで生成するデータである。κ_{QDC} は Energy deposit を MIP に変換するための変換係数で、σ_q は 1 MIP での TOF の Charge Resolution でキャリブレーションなどで決められ固

定されている。R_{gaus} はガウス分布をする乱数である。A は Attenuation Parameter で 0 から 1 の値をとる。実際に落としたエネルギーが PMT で検出されるまでに減衰する割合を表す。

実際は QDC の値から電荷 Q_{TOF} を求めるので、QDC のカウント値 QDC_{TOF} も算出する。

$$QDC_{TOF} = g \cdot Q_{TOF} \tag{6.13}$$

g は QDC のゲインで、1 MIP に対する QDC のカウント値である。

時間 T_{TOF} は

$$T_{TOF} = t \pm \frac{z}{v} + R_{gaus} \cdot \sigma_t \tag{6.14}$$

によって求める。t は時間で、シミュレーションから計算される値である。v は TOF 内部を粒子が走る速度、σ_t は時間分解能である。式に中の $\pm$ は TOF の両サイドの PMT を表す。また、実験データの場合は T は TDC の値から算出するものなので TDC の値も以下の式で算出する。

$$TDC_{TOF} = \frac{T_{TOF} + t_{\text{offset}}}{CLK} \tag{6.15}$$

t_{offset} は TDC の持つオフセット、CLK はクロック数である。

6.5.2 JET

JET では、FADC のカウント値 $FADC_{JET}$、Hit の z 位置 Z_{JET}[mm] ドリフト時間 T_{drift}[ns]、ドリフト距離 L_{drift}[mm] を求める。

$FADC_{JET}$ は以下の式で求める。

$$FADC_{JET} = E_{loss} \cdot \kappa_{FADC}(1 + R_{gaus} \cdot \sigma_q) \tag{6.16}$$

E_{loss}[keV] がシミュレーションから得られる Energy Deposit である。κ_{FADC} はエネルギーを FADC のカウント値に変換する変換係数である。また、実際には JET のワイヤーの両側から FADC で読み出しており、ヒットの z 位置によって両サイドで読み出す値は変化する。両サイドの FADC 値を $FADC_{JET0}, FADC_{JET1}$ とすると、

$$FADC_{JET0} = \frac{\epsilon}{\epsilon + 1} FADC_{JET} \tag{6.17}$$

$$FADC_{JET1} = \frac{1}{\epsilon + 1} FADC_{JET} \tag{6.18}$$

$$\epsilon = \frac{1 + z/q_{attn}}{1 - z/q_{attn}} \tag{6.19}$$

と求める。z[mm] がシミュレーションでの z 位置である。Q_{FADC0}, Q_{FADC1} はそれぞれワイヤーの両サイドからの FADC での読み出しである。同じ Hit でも読み出しまでの距離が違えば減衰する割合も変わってくるので、その違いを ϵ でかけ合せている。

ドリフト時間 T_{drift}、ドリフト距離 L_{drift} は以下の様に求める。

$$
\begin{aligned}
L_{\mathrm{drift}} &= dL + R_{gaus} \cdot \sigma_x && (6.20)\\
T_{\mathrm{drift}} &= \frac{L_{\mathrm{drift}}}{v_{\mathrm{drift}}} && (6.21)\\
dL &= \sqrt{(x - x_{wire})^2 + (y - y_{wire})^2} && (6.22)
\end{aligned}
$$

x, y[mm] がシミュレーションで得た位置情報、x_{wire}, y_{wire}[mm] は Hit のあったワイヤーの位置で、あらかじめ定義しておく。v_{drift}[mm/ns] はドリフト速度、σ_x は x 方向の位置分解能であらかじめキャリブレーションなどで決めておく。また、時間情報の電荷同様に FADC の値から算出するものなので、FADC の時間も求めておく。

$$
T_{FADC} = \frac{T_{drift} - T_{offset}}{CLK} \tag{6.23}
$$

T_{offset} は FADC の持つオフセットで、CLK はクロック数である。

Z_{JET} は

$$
Z_{JET} = z + \sigma_z \cdot R_{gaus} \tag{6.24}
$$

で求める。

6.5.3 IDC

IDC は基本的に JET と同じだが、真の Drift Length dL の求め方が異なる。

$$
\begin{aligned}
dL &= r_0 \cdot d\phi && (6.25)\\
d\phi &= |\phi - \phi_0| && (6.26)\\
\phi &= \arctan\left(-\frac{x}{y}\right) && (6.27)
\end{aligned}
$$

r_0, ϕ_0 は Hit のあったワイヤーの半径と角度である。ヒット位置 x, y よりその角度 ϕ を求め、ワイヤーとの角度の差から dL を求める。分解能の入れ方は JET と同じである。

6.5.4 ACC

ACC では電荷の QDC 値 QDC_{ACC} を求めるのだが、チェレンコフ光は粒子の速度があるしきい値以上にならないと発生しない。そのため、しきい値の上下で処理が少し異なる。

$1/beta > 1.02$ の場合はチェレンコフ光が発生するとみなし、

$$
QDC_{ACC} = \kappa_{QDC}\left(\lambda_{npe} + R_{gaus}\sqrt{\lambda_{npe}}\right) + R_{gaus} \cdot \sigma_q \tag{6.28}
$$

と求める。λ_{npe} は ACC の Npe の平均値、σ_q は QDC の Pedestal 分布の rms である。1.02 以下の場合は、Pedestal の分のみなので、式 6.28 の第 2 項のみを計算する。

6.6 True Track 情報の表示

これまで、BESS にはシミュレーション用のイベントディスプレイは存在せず、代わりに実験用のイベントディスプレイを使用していた。そのイベントディスプレイにはシミュレーションの True Track 情報を表示する機能はなく、Digitization も行った後のトラックとヒットを表示するだけであった。しかし、シミュレーションの議論のためには True Track 情報も表示する必要があった。そのために、新たにシミュレーションデータにも対応するイベントディスプレイを開発した。

6.6.1 データフォーマットの変更

True Track 情報を表示させるために、シミュレーションで出力されるバイナリファイルのデータフォーマットを変更した。表 6.8 にデータフォーマットを示す。

表 6.8: True Track 情報のバイナリフォーマット

パート	サイズ	フォーマット	データ	単位
Track Header	4	int	トラック数	
Track Data	2	int	Track ID	
	2	int	Parent Track ID	
	4	int	Particle ID	
	4	float	Vertex x	cm
	4	float	Vertex y	cm
	4	float	Vertex z	cm
	4	float	Momentum x	GeV/c
	4	float	Momentum y	GeV/c
	4	float	Momentum z	GeV/c
	4	float	Particle Energy	GeV
	4	float	Track Time	sec
	4	int	Step 数	

Track Header はそのイベント生成された全トラック数である。この数だけ以下の Track Data のパートが繰り返される。Track Data は各トラックのデータである。Track ID はそのトラックの ID、Parent Track ID はそのトラックを生成した親トラックの ID である。1 つ目のトラックはどのトラックからも生まれてないので、Parent Track ID は 0 である。Particle ID はそのトラックを構成する粒子の PDG コード、Vertex (x,y,z) はトラックの初期（生成された）位置、Momentum (x,y,z) はその粒子の初期運動量、Particle Energy は粒子の初期エネルギーである。Track Time は Track が生成された時間（イベント開始時を 0 とする）、Step 数はそのトラックを構成するステップの数である。

また、各ヒットがどのトラックに所属しているのか特定するために、Hit 情報に Track ID を追加した。

バイナリデータの変更に基づいて読み出しプログラムにも変更を加えたが、プログラムに下位互換性を持たせるためにファイルの先頭に表 6.9 バージョン情報を追加して管理することにした。GEANT 自体のバージョン情報も追加し、どの Version を用いたのかをわかるようにした。

さらに、GEANT3 を組んでいる FORTRAN はバイナリー形式で保存すると書き込んだデータのデータサイズを 4 byte 整数型で先頭と末尾に自動的に追加して書き込む。C++にはこの用な機能はないので、代わりにファイルの先頭のみに 4 byte に 0 を 1 度だけ書き込んでおく。ファイルの先頭 4 byte が 0 がそれ以外の値であるかで昔のデータとを区別し、0 であった場合は現在のデータなので、バージョン情報で見分ける。

表 6.9: File Header

パート	サイズ	タイプ	データ	単位
File Header	1	int	GEANT Version	
	1	int	GEANT Major Version	
	1	int	GEANT Minor Version	
	1	int	GEANT Patch Version	
	1	int	Program Major Version	
	1	int	Program Minor Version	

6.6.2 イベントディスプレイ

これまでのイベントディスプレイでは各ヒットと再構成されたトラックが表示されているが（図 6.4(左)）、表示されているトラックは JET, IDC, LTOF のヒットとは一致しても UTOF のヒットとは一致してない。データからはこれ以上の情報は得られないので、このイベントで何が起こっているのかは推測するしかない。もしくは、GEANT3 であればシミュレートした際の乱数のシードを保存しておき、確認したいイベントのシードを入力し、インタラクティブモードで起動して、コマンドで 1 つずつ確認するしかなかった。

True Track 情報を表示すれば全ての事象を見ることができるので、何が起こっているのかを全て確かめることができる。シミュレーションデータ用に開発したイベントディスプレイ「gevt」で先のイベントを表示した様子が図 6.4(右) である。実線が再構成されたトラック、破線が True Track、矢印がトラックの初期位置と運動方向を表している。さらに、True Track をクリックすればそのトラックの情報がコントロールパネルに表示される。

これを見れば、このイベントは入射した粒子が UTOF に Hit を作り、マグネット上部で反応し、2 次粒子が JET や LTOF に Hit を作っていることが分かる。この様に各イベントで起こっていることが確認できる他、解析プログラムは正しく機能し、トラックやヒットを再構成できているか確認することもできる。また、測定器内で

反応したイベントがバックグラウンドのイベントなっているかということの確認にも使用できる。

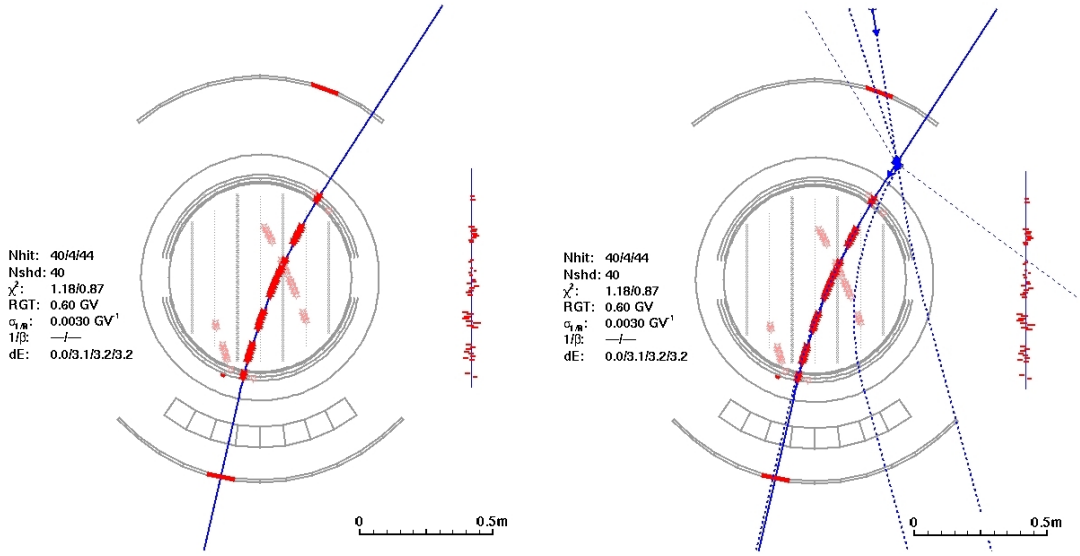

図 6.4: GEANT の True 情報を表示できるイベントディスプレイ「gevt」。以前のイベントディスプレイで表示できる情報だけではトラックと TOF のヒットが一致していない（左）。True Track 情報を表示すれば元々どういうイベントだったかが確認できる（右）。

第7章　評価

BESS のシミュレーターは、前回の BESS-PolarI 実験までのものは GEANT3 で構築されていて、低エネルギーの反陽子をシミュレートするために、Cross Section も調整されている。

g4lib の評価には、まず BESS-PolarI の Geometry を使い Geometrical Acceptance, Single Track Efficiency を g3lib と比較し使用している物理プロセスや Cross Section の妥当性を評価する。さらに、BESS-PolarI と BESS-PolarII との Geometry の違いによる Acceptance の変化を見積り、Geometry を BESS-PolarII にした時の Acceptance の変化を評価する。

7.1　検出器応答の再現

ここではまず Digitization Program で検出器応答が正しく観測データを再現できているかを確認する。

測定器の分解能は、観測データからは独自のフレークワークを用いて求める。分解能を含めて観測データを再現したシミュレーションデータにも、

シミュレーションの結果には、キャリブレーションで求めた分解能の値を入れるのだが、そのようにして再現したシミュレーションデータにもキャリブレーションをするときに用いたフレームワークと同様のものを用いて分解能を再計算して、観測データと比較する。

図 7.1 に JET の r-ϕ Resolution を示す。実線が観測データで赤のマークがシミュレーションデータを表す。r-ϕ Resolution とは、JET のヒット位置の x-y 方向の位置分解能である。x と y とを分けずに r-ϕ として一つの分解能を定義している。このグラフの広がりが分解能を表しており、観測データを正しく再現できていることが確認できる。

TOF の Timing Resolution を図 7.2 に示す。Timing Resolution とは TOF の時間分解能である。このグラフから Timing Resolution も観測データをよく再現していることが確認できた。

同様に他の検出器の応答についてもよく再現できている。

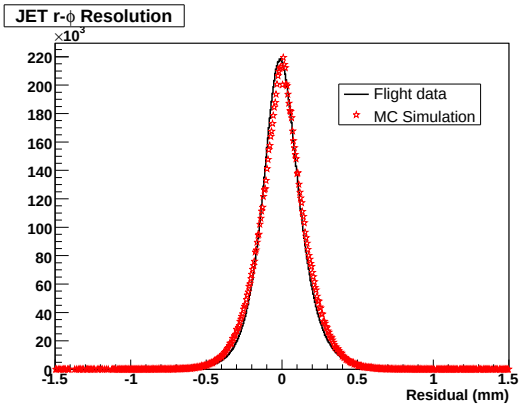

図 7.1: JET の r-ϕ Resolution

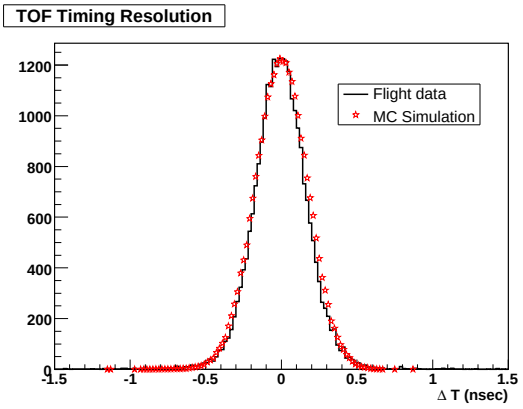

図 7.2: TOF の Timing Resolution

7.2 Acceptance の求め方

Acceptance $S\Omega$[m^{2}sr] は検出器の Geometry のみ決まるので以下の式で求める。

$$S\Omega = S\Omega_A \cdot \gamma_{fiducial} \tag{7.1}$$

$$S\Omega_A = \frac{1}{2} 4\pi \cdot \pi R_A^2 = 2\pi^2 R_A^2 \tag{7.2}$$

$$\gamma_{\rm fiducial} = \frac{n}{N} \tag{7.3}$$

まず、検出器の回りに半径 R_A の球 A を仮定する。その時の球 A の Geometrycal Acceptance が $S\Omega_A$ である。球 A の外から見ると球の領域は πR_A^2 見え、それらを全立体角で積分するために 4π をかける。しかし、BESS の検出器は上からの粒子しかとらえられないのでさらに 1/2 をかける。

$\gamma_{\rm fiducial}$ は球 A の表面の各点から降らせたイベントの内最終的にトラックとみなしたイベント数である。N は降らせたイベント数、n は最終的に track とみなしたイベント数である。本シミュレーションでは R_A は 2 m に固定している。

Geometrical Acceptance を求めるときには物理プロセスは使わないので、この値を算出するためには ALL-OFF モードのシミュレーション結果を用いる。Single Track Efficiency は全ての物理プロセスをシミュレートする必要があるので、ALL-ON モードのシミュレーションを用いる。

Acceptance を求めるときには、トラックが再構成できないイベントや Single Track にならかったイベントなど、使えないイベントを各種カットをかけて取り除く。以下に使用するカットを示す。図 7.3 は各カットをかける前後のカットパラメータの分布である。

1. トラック数（図 7.3, 1 行目 1 列目）

   Single Track のもののみを選択するので、トラック数 0 という再構成に失敗したものや、2 以上のものはカットする。ただし、トラック数が 2 でもどちらかがトラックと Fit してトラックの差が 100 mm 以内であればそれをトラックとして採用する。

2. UTOF の全ヒット数（図 7.3, 1 行目 2 列目）
   トリガーは UTOF+LTOF を使用するので、UTOF にヒットのないイベントはイベントとして成立しないのでカットする。また、ヒット数の多いイベントは多量の反応を起こしたノイズが多いイベントである可能性が高い。その様なイベントはトラックを再構成しにくく、分解能などの性能を下げるので、ヒット数が 3 以上のものはカットする。

3. LTOF の全ヒット数（図 7.3, 1 行目 3 列目）
   UTOF 同様にカットする。

4. UTOF のヒットのあったパドル数（図 7.3, 2 行目 1 列目）
   UTOF のヒット数と同様に、パドル数が 0 ということは UTOF からトリガーが出ていないということなので、その様なイベントはカットする。また、3 以上のものはノイズが多いとみなしカットする。

5. LTOF のヒットのあったパドル数（図 7.3, 2 行目 2 列目）
   UTOF 同様にカットする。

6. JET のヒット数（図 7.3, 2 行目 3 列目）
   JET のワイヤーは鉛直方向に多いところでも 40 本程度並んでいるだけである。ヒット数が多いということは、物質と反応を起こした量が多いと思われるので 100 を越えるイベントはカットする。

7. JET のセグメント数（図 7.3, 3 行目 1 列目）
   セグメントとはヒットのあった塊である。JET のヒット数同様、この量が多いということはノイズが多いと思われるので 15 を越えるイベントはカットする。

8. UTOF の x 位置のずれ（図 7.3, 3 行目 2 列目）
   TOF はパドルごとに位置が決まっている。ヒットの x 位置とパドルの x 位置 (中心位置) とのずれがパドルの幅をこえている場合、ヒット位置としては別のパドルをさしていることになる。その場合、トラックの再構成に失敗している可能性があるので TOF の幅（の半分）の 75 mm 以上のイベントはカットする。

9. LTOF の x 位置のずれ（図 7.3, 1 行目 3 列目）
   UTOF 同様にずれがパドルの幅より大きいイベントはカットする。

10. Rigidity（図 7.3, 4 行目 1 列目）
    トラックのずれ。計算に失敗して値が 0 になったものみにカットし、それ以外は特にカットしない。

11. χ^2_x（図 7.3, 4 行目 2 列目）
    トラックのずれ。計算に失敗して値が負（デフォルト値）になっているイベントのみをカットする。それ以外は特にカットしない。

12. Nshould（図 7.3, 4 行目 3 列目）

Nshould は JET ワイヤーのヒット数である。この値が小さい場合、検出器をかするような隅を通ったイベントである可能性が高く、トラックを再構成しにくいのでカットする。

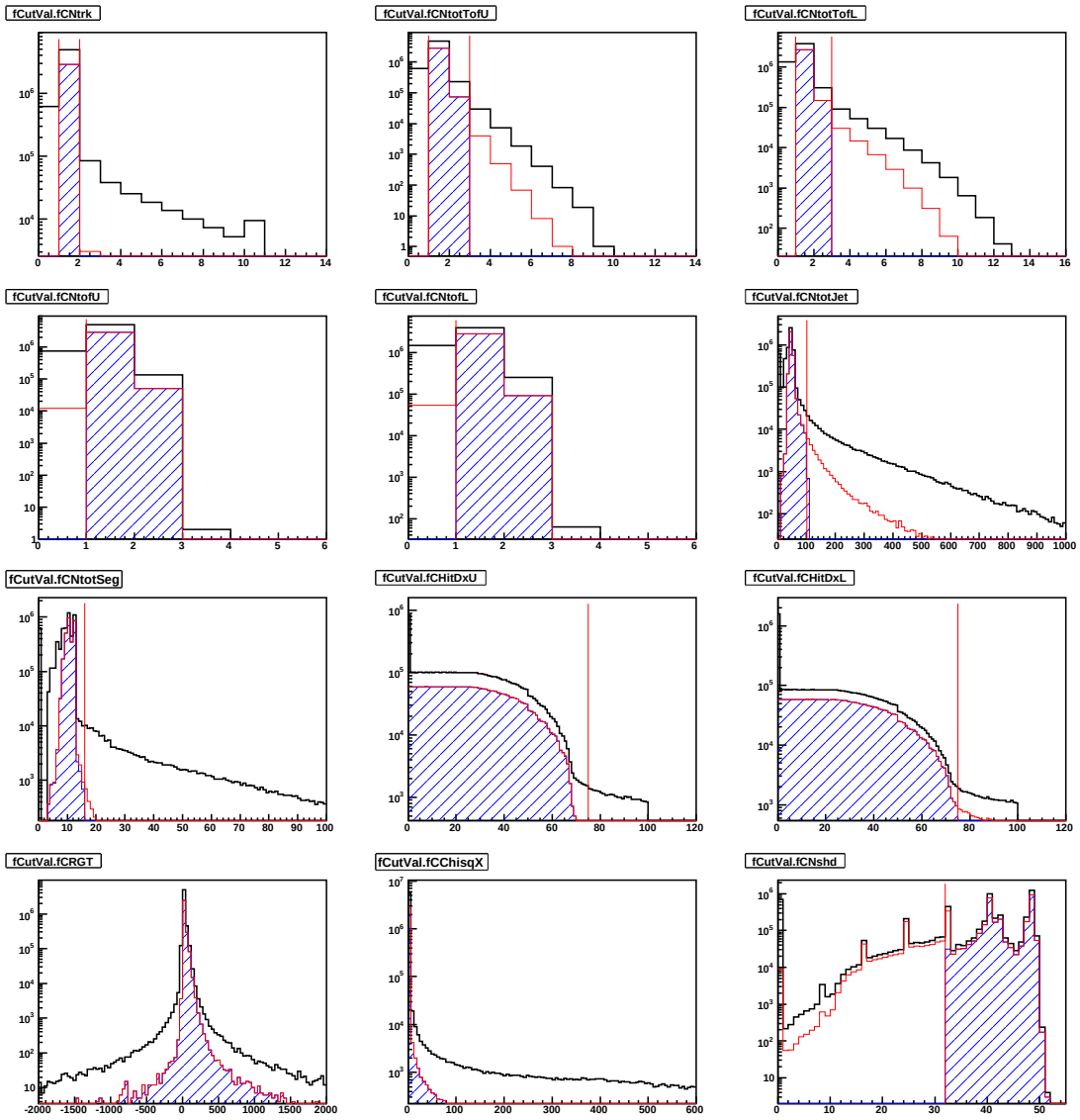

図 7.3: 各種カットパラメーターの分布。黒が何もカットをかけなかったときの分布。赤がそのカットを使わなかった時の分布。青（斜線）がカット後の分布。赤の縦線はカット位置。

7.3 Acceptanceの比較

7.3.1 Geometrical Acceptance

BESS-PolarI の Geometry を用いて求めた Geometrical Acceptance を g3lib と g4lib で比較する。図 7.4 に Proton の Geometrical Acceptace を比較した図を示す。

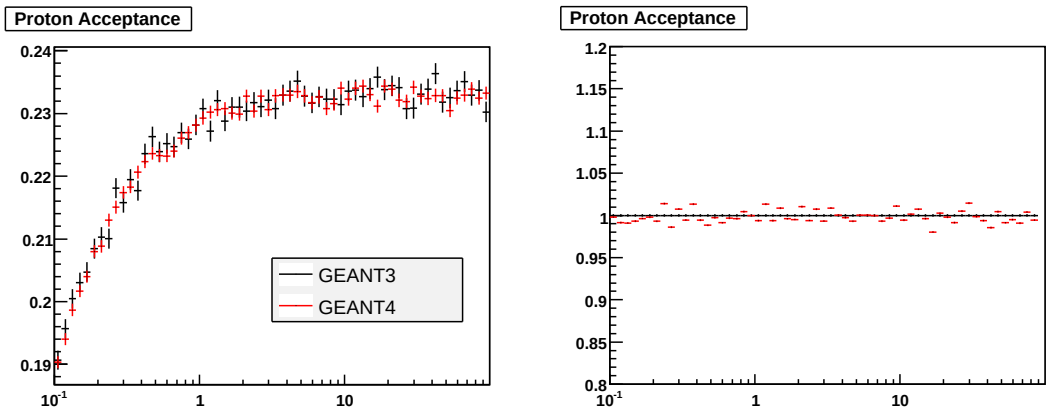

図 7.4: g3lib, g4lib の Geometrical Acceptance の比較 (BESS-PolarI Geometry)。Acceptance(左) と両者の比 (G4/G3)(右)。

2%の統計誤差の範囲内で一致している。

さらに、検出器は左右対称に作られているので、Anti-proton を使用した場合、磁場による軌跡が曲がる方向が左右対称であること以外 Proton と何も変わらないはずである。図 7.5 に Proton と Anti-proton での Gemetrical Acceptance を比較したものを示す。

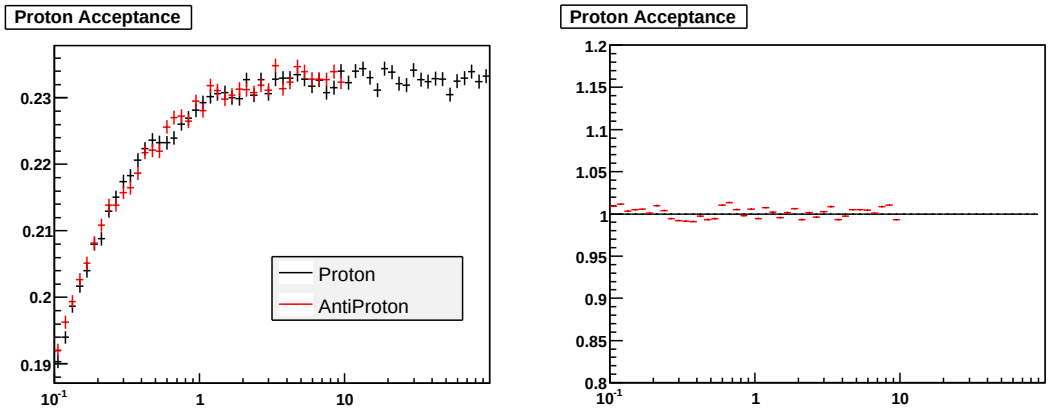

図 7.5: Proton と Anti-proton の Geometrical Acceptance の比較

Proton と Anti-proton の Geometrical Acceptance は一致している。

これらのことから、g4lib には BESS-PolarI の Geometry が正しく入っていることが確認できた。

7.3.2 Single Track Efficiency

Single Track Efficiency の算出には ALL-ON モードを用いる。このモードでは全ての物理プロセスが有効になっているので、検出器の geometry だけでなく、Cross Section 等も影響してくる。検出器内部で様々な反応が起こり、2 次粒子が生まれる。例えば、本来 UTOF しか通らないイベントでも、途中で 2 次粒子が生まれることによって、2 次粒子が MTOF, LTOF に Hit してイベントとなることがある。その様なイベントも含めて、Single Track として認識できる割合を算出できる。

図 7.6 に g3lib, g4lib の陽子の Single Track Efficiency を比較したものを示す。5%の範囲で一致しているが、数% GEANT4 が系統的に低く見える。

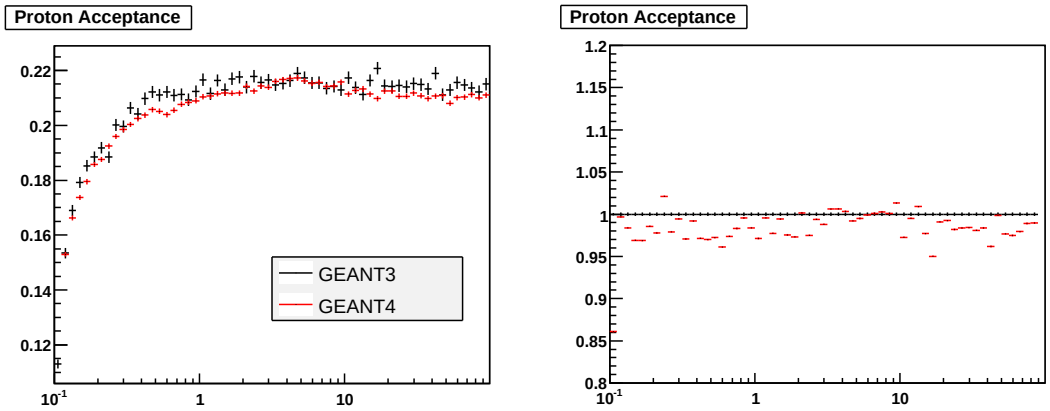

図 7.6: g3lib, g4lib の Single Track Efficiency の比較。Efficiency の図 (右) と両者の比 (G4/G3) の図 (左)。

陽子はトリガー条件として UTOF+LTOF のものして使っていなかった。しかし、反陽子は LTOF に到達できないような低エネルギーの粒子も検討しなければならない。よって、UTOF+LTOF と UTOF+MTOF の両方を評価する。図 7.7 に反陽子の UL トリガーでの Effiency、図 7.8 に UM トリガーでの Efficiency を示す。

Anti-proton の Single Track Efficiency は UM で一致している。しかし、UL は G4 の方が系統的に数%低い。次からはこの違いについて考察する。

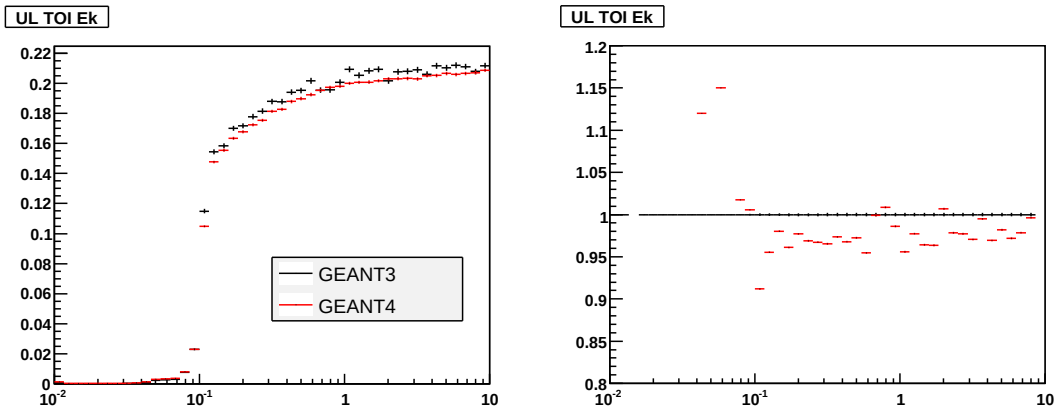

図 7.7: g3lib と g4lib の反陽子（UL）の Single Track Efficiency の比較。Efficiency（左）とその両者の比（G4/G3）（右）

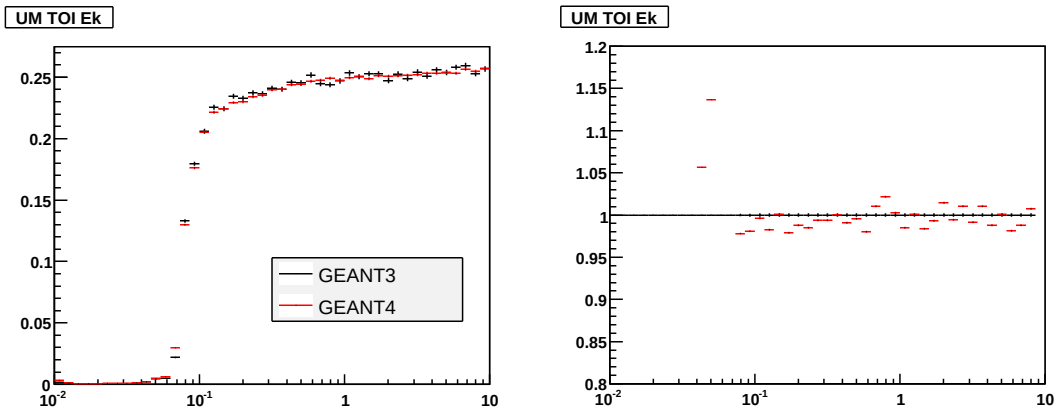

図 7.8: g3lib と g4lib の反陽子（UM）の Single Track Efficiency の比較。Efficiency（左）とその両者の比（G4/G3）（右）

7.4 Tracking Cut による違いの考察

陽子の Single Track Efficiency、反陽子 UL の Single Track Efficiency が G4 の方が数%系統的に低い。この違いを考察するために、GEANT3 と GEANT4 の違いである Tracking Cut について考える。

7.4.1 Event Reduction

Single Track Efficiency は元々のイベント数から残ったイベント数の比率なので、一致していないということは残ったイベント数がずれているはずである。そのため、各カットごとにイベントのカットされる量を比較して、どのカットで違いが出ているのかを確認する。イベントのカットされる様子として Event Reduction を図 7.9 に示す。カット番号の意味は 7.2 とは一致しないので表 7.1 で改めて説明する。

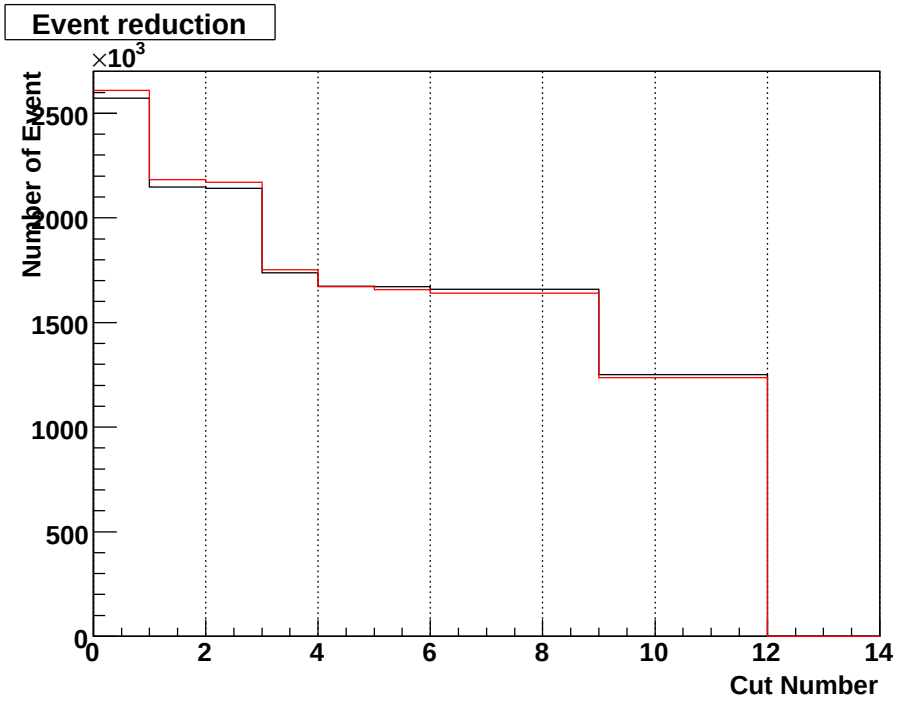

図 7.9: Event Reduction。横軸はカット番号である。$x = 0$ の時がカット前のイベント数でカットかける度にイベント数が減っていく様子を表している。

Event Reduction で G3/G4 で違いがある部分として、

- カット前のイベント数が G4 が少し多い
- カット#4 で G4 が多くカットされている
- カット#5 で G4 が多くカットされている

表 7.1: 図 7.9 の各カットの内容

カット番号	カット	カッとされるイベント
1	トラック数	トラック数が 1 でない (一部 2 も残る)
2	UTOF ヒット数	UTOF の全ヒット数が 1, 2 でない
3	LTOF ヒット数	LTOF の全ヒット数が 1, 2 でない
4	トラック内の TOF ヒット数	再構成されたトラックに使用されてる UTOF か LTOF のヒット数が 0
5	JET ヒット数	JET のヒット数が 100 を超えるか セグメント数が 15 を超える
6	TOF dx	UTOF か LTOF のヒット位置 (x) とパドルの 中心位置のずれ (dx) が 75 mm を超える
7	Rigidity	Rigidity = 0
8	χ^2_x	$\chi^2_x \leq 0$
9	Nshould	Nshould が 32 未満か
10	TOF PMT マスク 1	(使用しない)
11	TOF PMT マスク 2	(使用しない)

が考えられる。カット#5 は JET のヒット数のカットなので、JET のヒット数が多いと思われる。また、カット#4 からも再構成されたトラックと TOF のヒットが一致していないようなイベントが多いことがわかるので、トラックが正しく再構成できないほど JET に余計なヒットが多いと思われる。また、カット前のイベント数が多いので、TOF のヒットも多いと思われる。JET のヒット数分布の比較を図 7.10 に、U,M,LTOF のパドルごとのヒット数を図 7.11 に示す。

図 7.10 の JET のヒット数分布はヒット数 50 以上の領域でテールの形が食い違っている。また、図 7.11 からも G4 の方が系統的に多い。

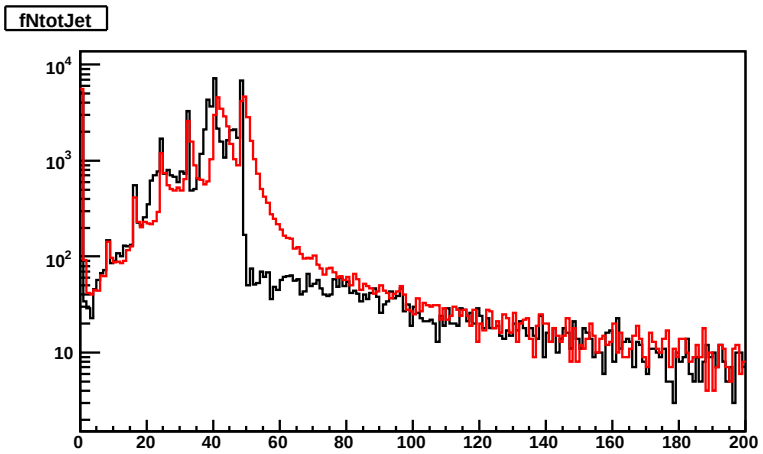

図 7.10: ALL-ON モードでの JET のヒット数の比較

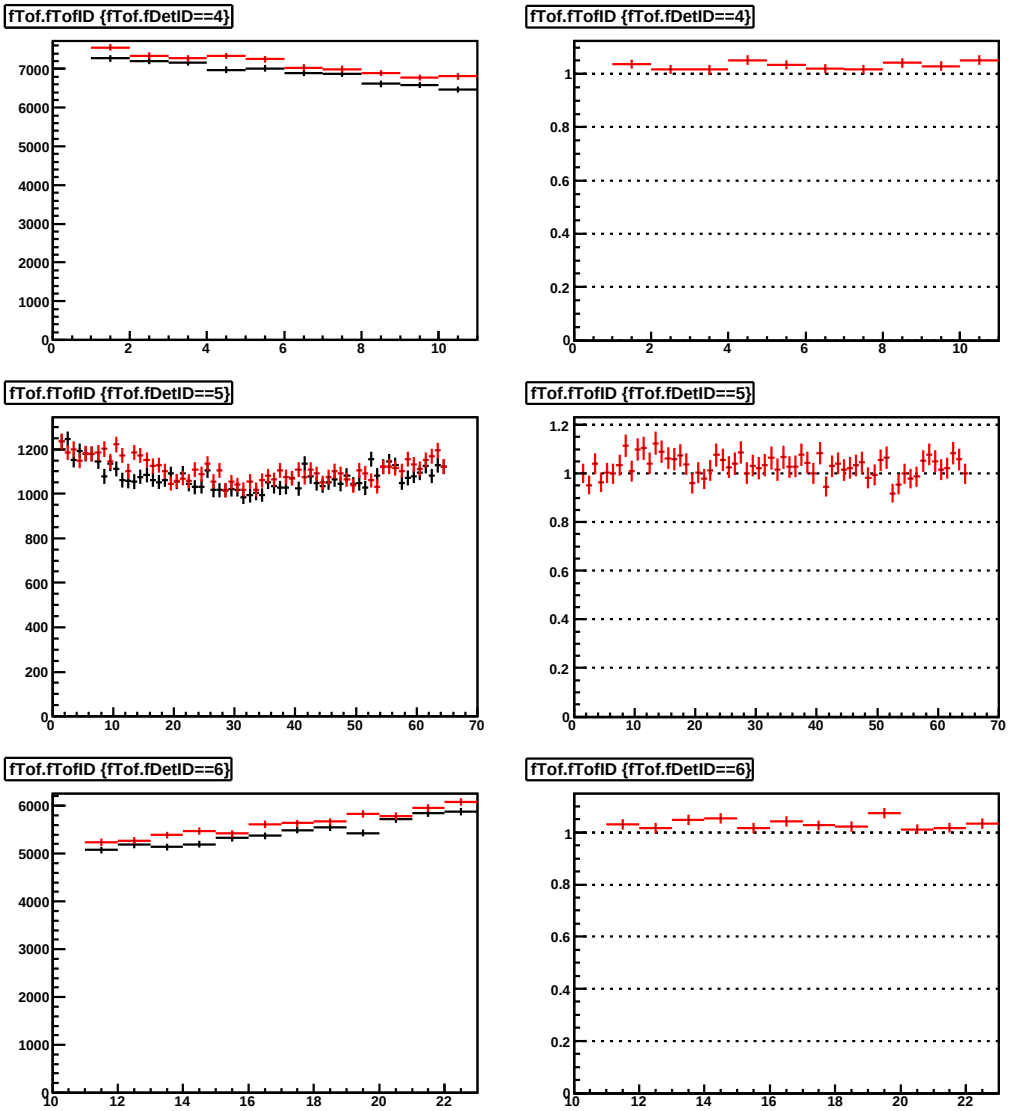

図 7.11: ALL-ON モードでの TOF の paddle ごとのヒット数の比較。上から順に UTOF(上), MTOF(中), LTOF(下) の分布で。左はヒット数の比較、右は両者の比率

7.4.2 Tracking Cut

ここで、GEANT3 と GEANT4 の構造の違いについて考える。両者の違いとして、エネルギー損失の計算方法とそれに伴う Tracking Cut がある。

GEANT3 ではエネルギー損失は dE/dx がステップ中で一定と仮定して、

$$\Delta E = \left\langle \frac{dE}{dx} \right\rangle L_{step} \tag{7.4}$$

で計算している。1 ステップのエネルギー変化割合に制限があり、エネルギーが低くなるとステップ長が極端に短くなるため、粒子のエネルギーを 0 まで追えない。そのため、あるエネルギー以下になると粒子を追うのをやめる。そのエネルギーのしきい値を Tracking Cut というもので設定されている。

表 7.2 に g3lib で使用していた Tracking Cut の値を示す。低エネルギーの反陽子を解析するために、荷電粒子用のパラメーターは GEANT3 デフォルトの値より低く設定してある。

表 7.2: Tracking Cut の各パラメーターの値

適応する粒子	値 [MeV]
γ	1.0
e^+/e^-	1.0
μ	1.0
Neutral Hadrons	10.0
Charged Hadrons	0.2

これに対し、GEANT4 でのエネルギー損失の計算方法が見直されて、積分方法を用いるようになった。

$$\Delta E = \int_0^{L_{step}} dx \frac{dE}{dx} E(x) \tag{7.5}$$

これによって、ステップ長の制限を与えることなく、エネルギー 0 まで粒子を追えるようになった。

Tracking Cut があるために、GEANT3 ではあるエネルギー以下の粒子は追われず、生成された粒子が初めからカット値以下だった場合全く Tracking されない。そのためにヒット数が GEANT4 より少なかったのだと思われる。g4lib にも表 7.2 の Tracking Cut を導入してシミュレートしてみる。

Tracking Cut 導入後の JET のヒット数分布を図 7.12 に、TOF のパドルごとのヒット数分布を図 7.13 に示す。JET のヒット数 50 以上にあったテールのずれがなくなった。TOF のヒット数に関しても系統的ずれがほぼなくなり、5%以内で一致している。

JET, TOF のずれが解消されたので、改めて Event Reduction を図 7.14 に比較する。Tracking Cut を導入したことにより、δ-ray などのエネルギーの非常に低い粒子が減りヒット数が減った。そのため、JET/TOF のヒット数によってカットされるイベントが減り、G3/G4 でより一致した。

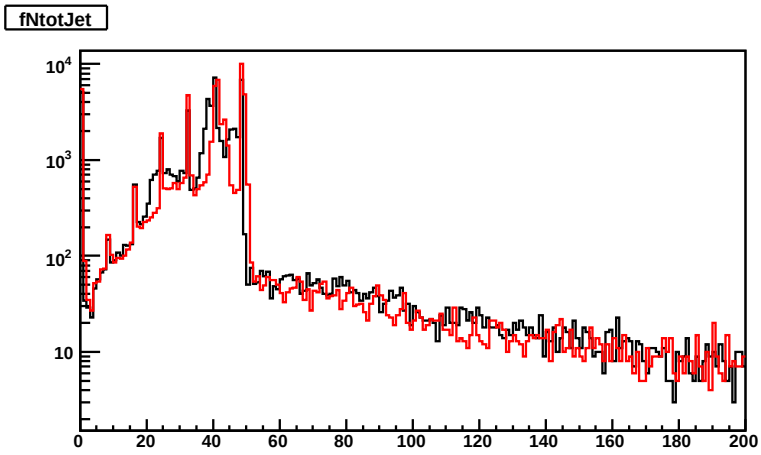

図 7.12: ALL-ON モードでの JET のヒット数の比較。（Traking Cut 導入後）

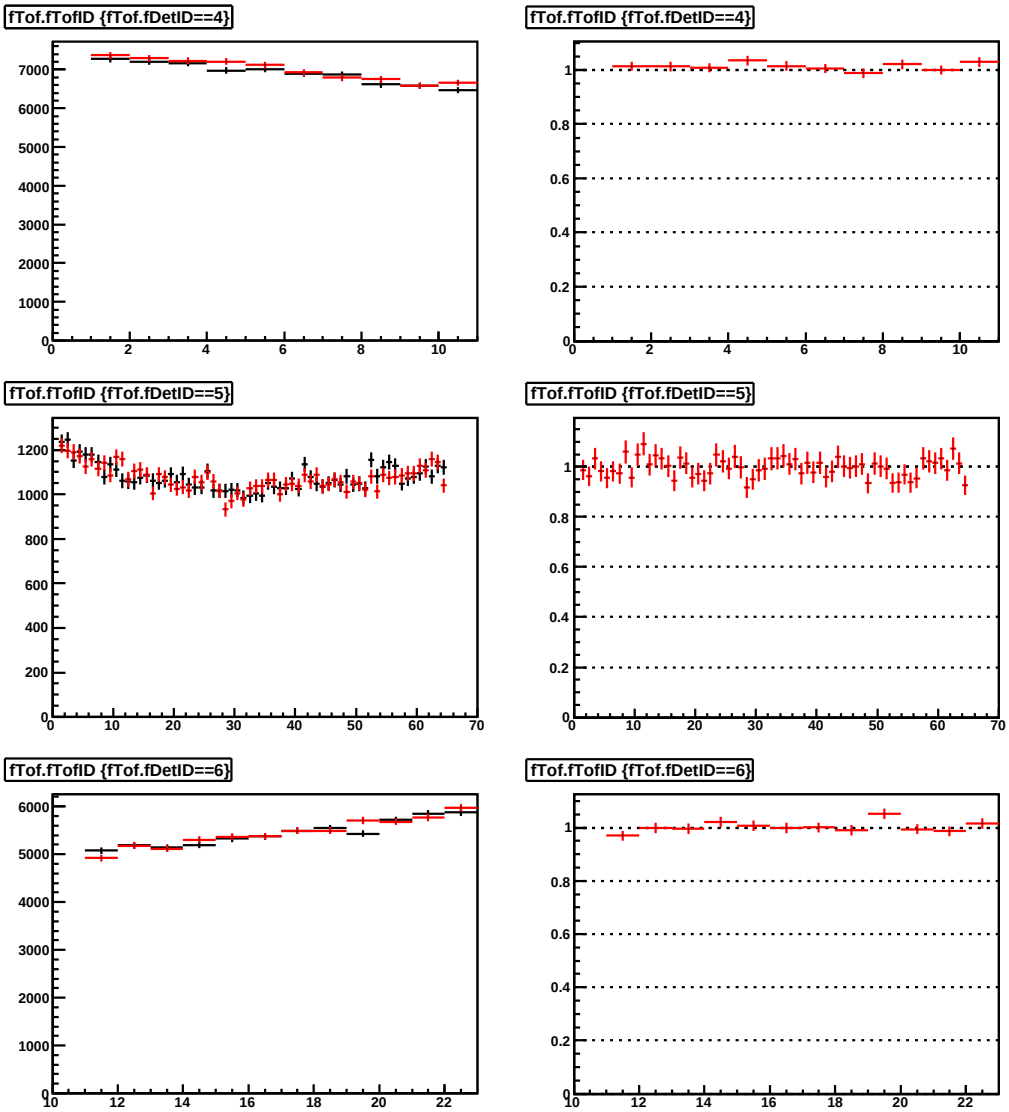

図 7.13: ALL-ON モードでの TOF の paddle ごとのヒット数の比較（Tracking Cut 導入後)。左はヒット数の比較、右は両者の比率

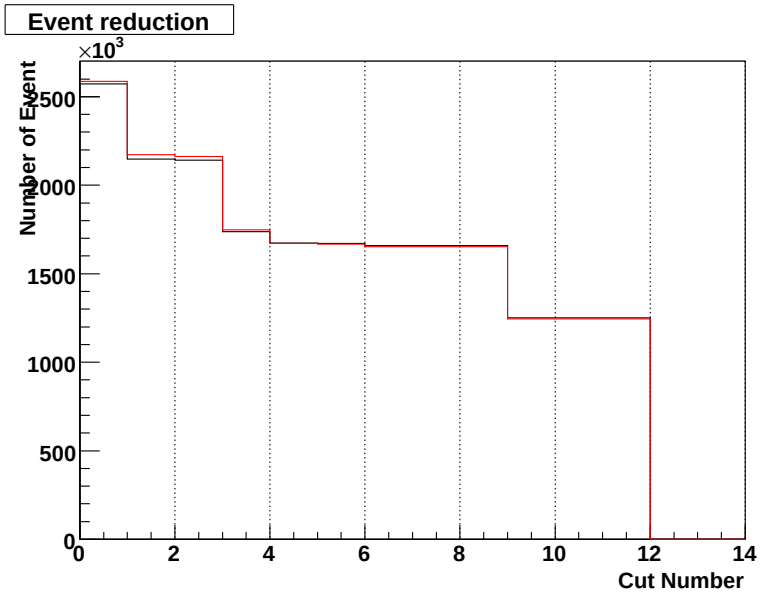

図 7.14: Event Recution（Tracking Cut 導入後）

このシミュレーションデータから求めた Single Track Efficiecy を、陽子を図 7.15 に、反陽子の UL を図 7.16、UM を図 7.17 に示す。反陽子の UL はまだ少しずれが見えるようにみえるが、Tracking Cut 導入前にあったずれはほぼなくなり、陽子は 2%以内、反陽子は UM は 2%以内、UL も 3%以内で一致した。

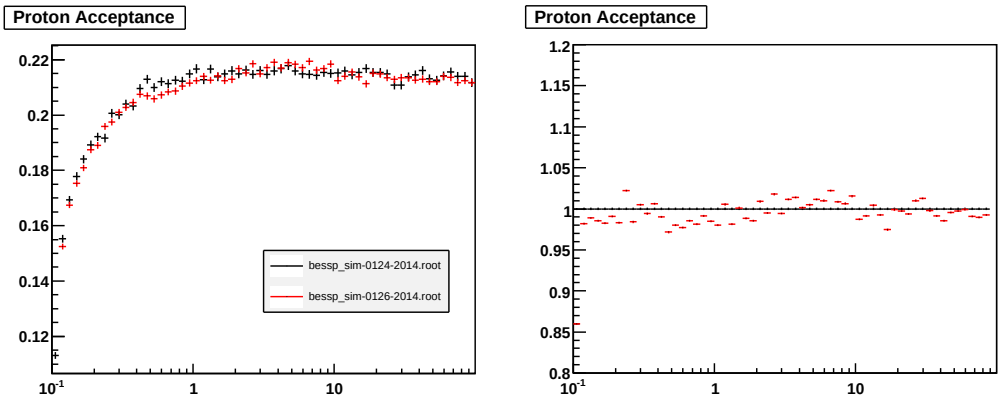

図 7.15: Tracking Cut 導入後の ALL-ON の Proton Single Track Efficency

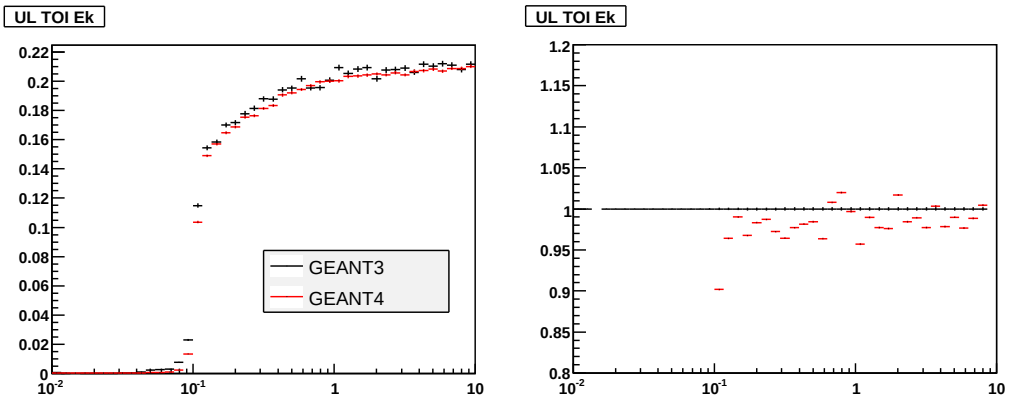

図 7.16: Tracking Cut 導入後の ALL-ON(UL) の Anti-Proton Single Track Efficiency

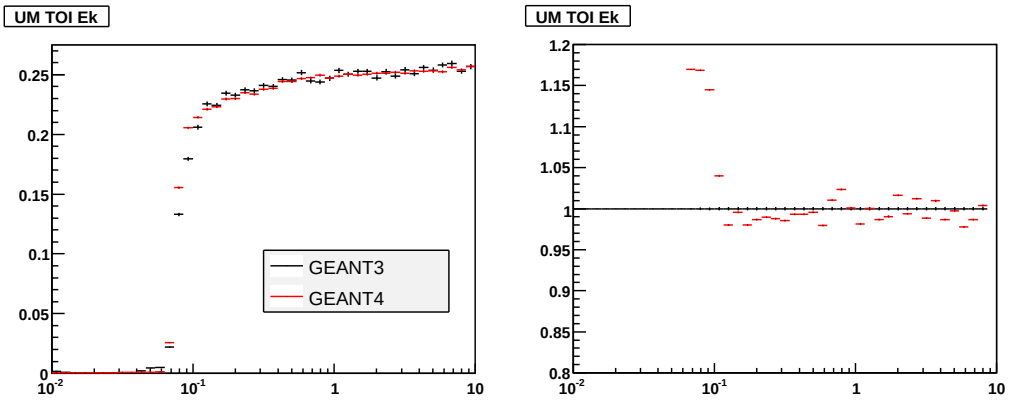

図 7.17: Tracking Cut 導入後の ALL-ON(UM) の Anti-Proton Single Track Efficiency

7.5 PolarIIへの移行

7.5.1 Geometrical Acceptanceの変化

BESS-PolarIのデータが正しく移植できたことが確かめられたので、次はGeometryをBESS-PolarIIのものにして検証する。

まず、Geometryが正しく組み込めているかを評価するために、Geometrycal Acceptanceを用いる。BESS-PolarIIのデータは、BESS-PolarIの様に比較する対象が存在しない。そこで、Geometrical AcceptanceのBESS-PolarIからの変化を評価する。

Geometrical Acceptanceは検出器の構造のみで決定されるので、検出器の構造の変化を知っていれば、Geometrical Acceptanceがどの程度変化するはずなのかを見積もることができる。実際のAcceptanceの変化が見積もった変化分と一致するかを確かめる。

検出器のGeometryは主にTOF, ACCについて変更がなされた。Geometrical AcceptanceはTOFを通るかどうかのみに影響するので、TOFの変更内容を表7.3に示す。

表 7.3: BESS-PolarIからBESS-PolarIIへのTOFの構造の主な変更点

検出器	パート	BESS-PolarI	BESS-PolarII
UTOF	UTOFが覆う角度	81.95°	85.03°
	UTOFの曲率半径	73.5 cm	73.6 cm
LTOF	LTOFが覆う角度	95.574°	92.56°
	LTOFの曲率半径	74.5 cm	78.6 cm
MTOF	MTOFが覆う角度	90.986°	91.69°
	MTOFの曲率半径	39.67 cm	39.645 cm
	MTOF Scintillatorの長さ	95.0 cm	100.0 cm

Acceptanceの変化分は簡易モデルを用いて計算した。簡易モデルでは磁場は使用せず直線のみで計算した。低エネルギー領域では軌跡は大きく曲がるので、必ずしもこの計算と一致はしないが、高エネルギー側では粒子はほとんど曲がらないと仮定してこの結果を適用してみる。

簡易モデルでの計算の結果、PolarIからPolarIIへのGeometryの変化によって、Acceptanceは1.012倍になることがわかった。PolarIのGeometrical Acceptanceに変化分をかけ合せたものとPolarIIのGeometrical Acceptanceとの比較を図7.18に示す。左のグラフは、黒がPolarI Geometryでの Geometrical Acceptance、赤がPolarII Geometryでの Geometrical Acceptance、そして、青がPolarIのGeometrical Acceptanceに簡易モデルで算出した1.012倍をかけ合せたPolarIIの予測Acceptanceである。PolarIIのAcceptanceと、PolarIから予測したPolarIIのAcceptanceとの比率が右のグラフである。曲がり具合の大きい低エネルギー側ではずれがあるが、1 GeV以上の高エネルギー側では予測どおりの値となっており、PolarIIのGeometryも正しく入れられていることが確認できた。

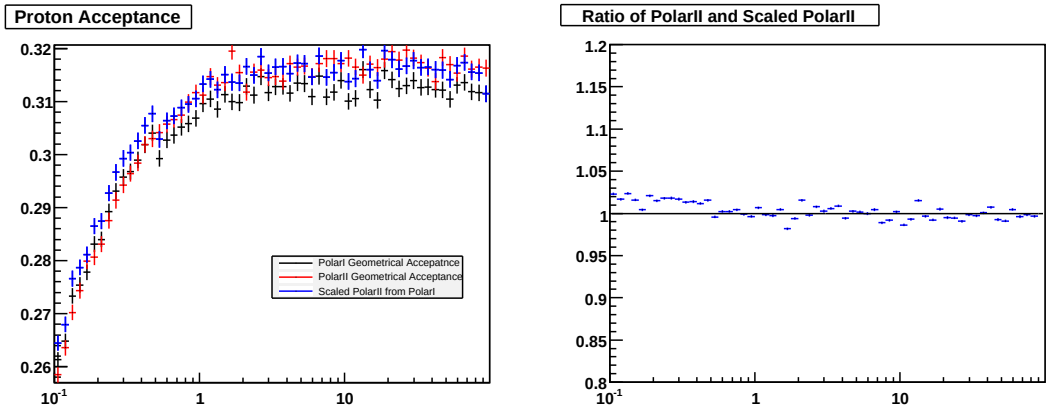

図 7.18: PolarI の Geometrical Acceptance から予測される PolarII の Geometrical Acceptance（左図青）と、PolarII の Geometry でシミュレートした Geometrical Acceptance（左図赤）の比較

7.6 まとめと今後の課題

本研究ではそれまで GEANT3 で開発されてきた BESS のシミュレーターを GEANT4 に移植し、BESS-PolarII 用に開発した。BESS-PolarII の Geometry を入力し、GEANT3 でのシミュレーションで調整されていた Cross Section も移植し、検出器応答が正しく行われていることも確認でき、BESS-PolarII から GEANT4 でのシミュレーションを使い解析をする準備ができた。

今後は Digitization のパラメーターや検出器応答の計算の仕方などを調整して、より実データとあうようにプログラムを改良する必要がある。

第8章　まとめ

BESS 実験では宇宙反陽子の探索を目的として 2 度の南極観測を行った。2 度目の南極実験である BESS-PolarII 実験は 2007 年 12 月に実施され、BESS-PolarI での不具合を改善・改良し約 24.5 月間におよぶ長期間の太陽活動極小期における宇宙線を観測することに成功した。本研究では、主に DAQ を担当し、FADC 接続の改良、USB デバイスドライバーの改良、通信データ・オペレーションコマンドの改良、自動化に関する改良などを行い、非常に安定した高速な DAQ を実現することに成功した。

また、BESS-PolarI までの解析では、測定器の Acceptance の評価に GEANT3 を用いて開発されていた。しかし、BESS-PolarII のためのシミュレーターは開発されておらず、また、FORTRAN ベースの GEANT3 は技術者がメンテナンスが終了しており、FORTRAN 技術者も減少してきている。そのため、BESS-PolarII では GEANT4 を用いて測定器シミュレーターの開発を行った。BESS-PolarI のシミュレーターでは陽子・反陽子の Cross Section が調整されているのでそれらを含めた物理プロセスを移植し、BESS-PolarII 測定器の Geometry を入力した。Acceptance 等で GEANT3 のシミュレーションとの整合性を確かめて、BESS-PolarII で反陽子解析に使用する準備が整った。

第9章　謝辞

指導教官である武田廣教授および川越清以教授、野崎光昭元教授には様々な御指導を頂き、深く感謝致します。研究を進める上でデータ収集系、ハードウェア、ソフトウェアにおける様々な指導をしてくださいました佐々木誠氏、そして実験装置から実験手法に至る詳細を教えていただきました山本明教授、吉村浩司准教授、灰野貞一氏、松田晋弥氏、長谷川雅也氏、坂井賢一氏、堀越篤氏には大変御世話になりました。さらに蔵重久弥准教授、原俊雄准教授をはじめとする神戸大学の皆様にも深く御礼を申し上げます。また海外出張で不在が多く様々な事務処理をお願いしました横山有美さんに感謝致します。その他のグループ内外を問わず、ここに挙げられなかった皆様に感謝の意を表したく思います。本当にありがとうございました。

付 録A　g4libクラス構造

ここではg4libのクラス構造および各クラスの機能・役割について説明する。なお、クラス構造のモデリングにはUMLを用いている。

A.1 Geometry

A.1.1 Detector Construction

Detector Constrctionのクラス図を図A.1に示す。

PDetectorConstruction Geometry定義クラス

PDetectorContructionクラスはGEANT4のG4VUserDetectorConstructionを継承している。このクラスでは全Geometryとそれに使用するMaterialを定義する。Detectorは全てPDetctorPartsを基本クラスとして定義し、Detectorの作成は全てPDetectorFactoryに移譲する。PDetectorFactoryは各DetectorをPDetectorPartsとして生成する。Materialの生成は全てMaterialFactoryに移譲し、Material FactoryはPDetectorConstructoinのコンストラクタで渡しておく。

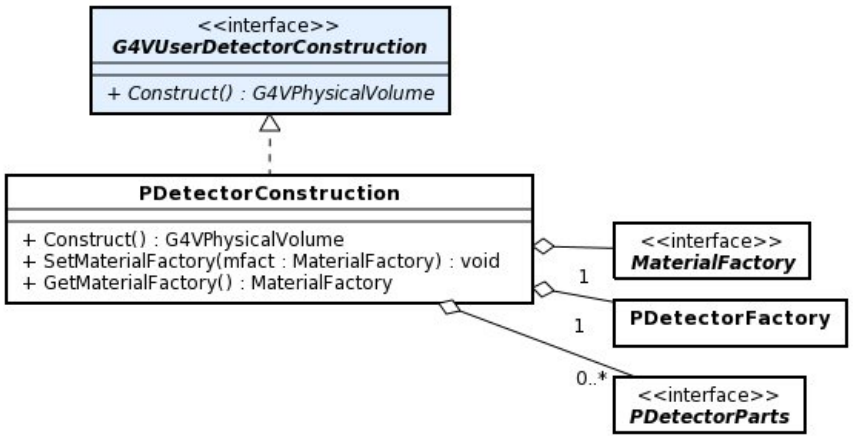

図 A.1: Detector Constructionのクラス図

A.1.2 Detector Factory

Detector Factory のクラス図を図 A.2 に示す。

PDetectorFactory Geometry 生成クラス

PDetectorFactory は指定されたタイプの Geometry を生成する。タイプは、BESS-PolarI が「P1」、BESS-PolarII が「P2」である。このクラスはシングルトンとして作成されているため、直接インスタンスは生成できないので、PDetectorFactory::GetFactory 関数で生成する。また、その際に引数にタイプを指定する。デフォルトは P1 である。

Detector は各 Create 関数で生成され PDetectorParts として返される。引数には生成する Detector が所属する Mother Volume の Logical Volume と Material Factory を指定する。CreateWorld 関数だけはトップレベルの Geometry なので Mother Volume を指定する必要はない。CreateMagneticField 関数で一定領域に磁場がかけられる。

Tracking Cut の様な Limit を指定するときは、SetLimits 関数および SetLimitsAll 関数で設定する。設定したい Volume の Logical Volume と、G4UserLimits クラスを継承した Limit 用クラスを引数に指定する。SetLimitsAll 関数を使用した場合、指定した Logical Volume が持つすべての子 Volume に Limit が適用される。

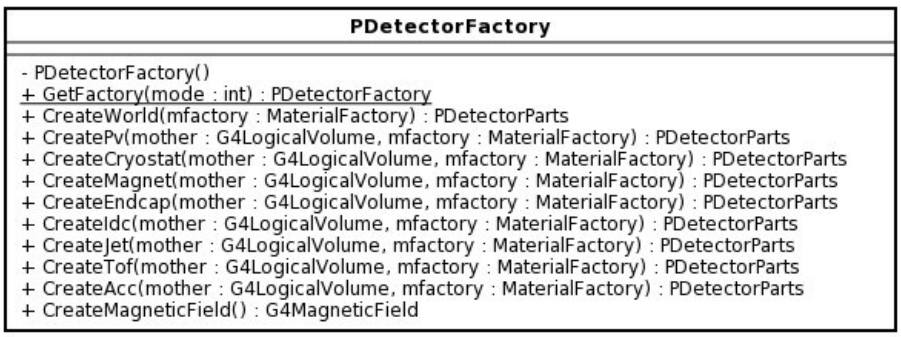

図 A.2: Detector Factory のクラス図

A.1.3 Material Factory

Material Factory のクラス図を図 A.3 に示す。

MaterialFactory MaterialFactory の基本クラス
Mateiral Factory のクラスはこのクラスを継承する。

この派生クラスでは、DefineElements 関数で各種 Element を定義し、DefineMaterials 関数で各種 Material を定義する。定義した Element, Material は AddElement 関数、AddMateiral 関数でリストに登録する。登録した Element, Material は GetElement 関数、GetMaterial 関数で取得する。Element は表 A.1、Material は表 A.2 のものが指定できる。

表 A.1: Element List

No.	Label	Element
1	E_C	C
2	E_H	H
3	E_N	N
4	E_O	O
5	E_NB	Nb
6	E_TI	Ti
7	E_CU	Cu

表 A.2: Material List

No.	Label	Material
1	M_VACUUM	Vacuum
2	M_AIR	Air
3	M_ALUMINIUM	Aluminium
4	M_COPPER	Copper
5	M_MYLAR	Mylar
6	M_MAG_CONDUCTOR	Mag Conductor
7	M_ZYLON	Zylon
8	M_ROHACELL	Rohacell
9	M_SCINTILATOR	Plastic Scintilator(CH)
10	M_IDC_GAS	IDC Gas (CO_2)
11	M_GFRPG10	GFRP G10
12	M_GLUE	Glue
13	M_AIREXULTEM	Air Exultem
14	M_CARBONFORM	Carbonform
15	M_GATHERFORM	Gatherform
16	M_POLYTHELENE	Polythelene
17	M_AGEL	Aerogel

G3MaterialFactory GEANT3 の Material を生成する Factory クラス
g3lib で使用した Mateial と同様の Material を生成する Factory クラス。

G4MaterialFactory GEANT4 の Material を生成する Factory クラス
GEANT4 であらかじめ定義されている Element を使用して Material を定義する Factory クラス。

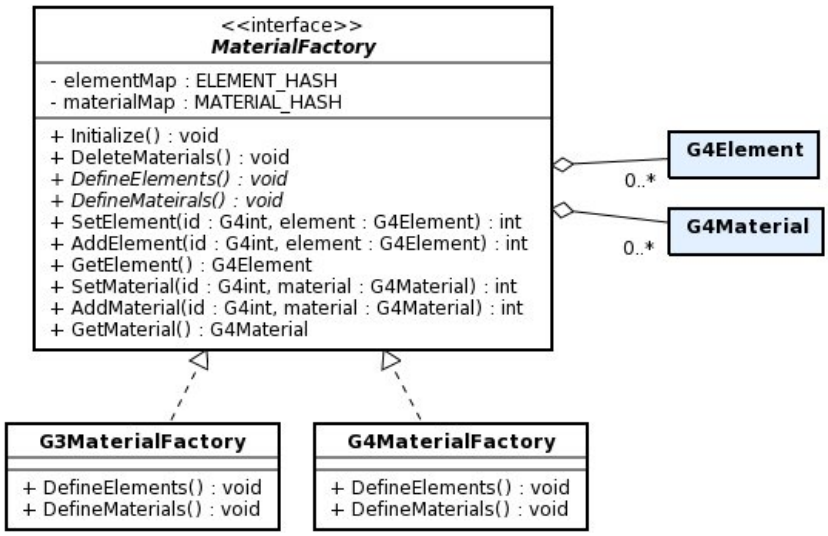

図 A.3: Material Factory のクラス図

A.1.4 Detector Parts

Detector Parts のクラス図を図 A.4 に示す。PDetectorParts もしくは PSDetector-Parts を継承した PolarI 用 Geometry クラスは名前空間 P1Detector 内で定義されている。PolarII 用 Geometry クラスは Geometry に変更のあった Tof, TofPad, Mtof, MtofPad, Acc, AccBlock のみ名前空間 P2Detector 内で定義されており、それ以外は PolarI のクラスを使用する。

PDetectorParts Detector の基本クラス
メンバ変数に Detector として定義するのに必要な、G4VSolid, G4LogicalVolume, G4VPhysicalVolume を持っている。派生クラスは、Initialize 関数をオーバーライドし各種定義を行う。　全ての Detector はこのクラスを継承する。

PSDetectorParts Sensitive Detector の基本クラス
PDetectorParts に Sensitive Detector として定義するのに必要な G4VSensitiveDetector を付け加えたクラス。全ての Sensitive Detector はこのクラスを継承する。

World World の Geometry
World Volume の Geometry クラス。PDetectorParts クラスを継承している。

Endcap Endcap の Geometry
Endcap の Geometry クラス。PDetectorParts クラスを継承している。

Cryostat Cryostat の Geometry
Cryostat の Geometry クラス。PDetectorParts クラスを継承している。

Magnet 超伝導マグネットの Geometry
超伝導マグネットの Geometry クラス。PDetectorParts クラスを継承している。

PressureVessel Pressure Vessel の Geometry

Pressure Vessel の Geometry クラス。PDetectorParts クラスを継承している。

Tof TOF の Geometry

TOF(UTOF+LTOF) の Geometry クラス。内部で UTof, LTof を生成する。PDetectorParts クラスを継承している。

UTof UTOF の Geometry

UTOF 全体の Geometry クラス。内部で TofPad を生成する。PDetectorParts クラスを継承している。

LTof LTOF の Geometry

LTOF 全体の Geometry クラス。内部で TofPad を生成する。PDetectorParts クラスを継承している。

TofPad TOF Paddle の Geometry

TOF Paddle の Geometry クラス。Sensitive Detector なので PSDetectorParts クラスを継承している。

MTof MTOF の Geometry

MTOF 全体の Geometry クラス。内部で MTofScinti を生成する。PDetectorParts クラスを継承している。

MTofScinti MTOF Scintillator の Geometry

MTOF Scintillator の Geometry クラス。Sensitive Detector なので PSDetectorParts クラスを継承している。

Acc ACC の Geometry

ACC 全体の Geometry クラス。内部で ACC Block を生成する。PDetectorParts クラスを継承している。

AccBlock ACC Block の Geometry

ACC の各ブロックの Geometry クラス。Sensitive Detector なので PSDetectorParts クラスを継承している。

Jet JET の Geometry

JET の Geometry クラス。内部で JetD クラスを生成している。PDetectorParts クラスを継承している。

JetD JET Wire の Geometry

JET の Wire の Geometry クラス。Wire のある領域を Box として定義する。Sensitive Detector なので PSDetectorParts クラスを継承している。

Idc IDC の Geometry

IDC の Geometry クラス。内部で IdcPlane を生成する。PDetectorParts クラスを継承している。

IdcPlane IDC Plane の Geometry

IDC Plane の Geometry クラス。内部で IdcPad を生成する。PDetectorParts クラスを継承している。

IdcPad IDC Pad の Geometry

IDC Pad の Geometry クラス。Sensitive Detector なので PSDetectorParts クラスを継承している。

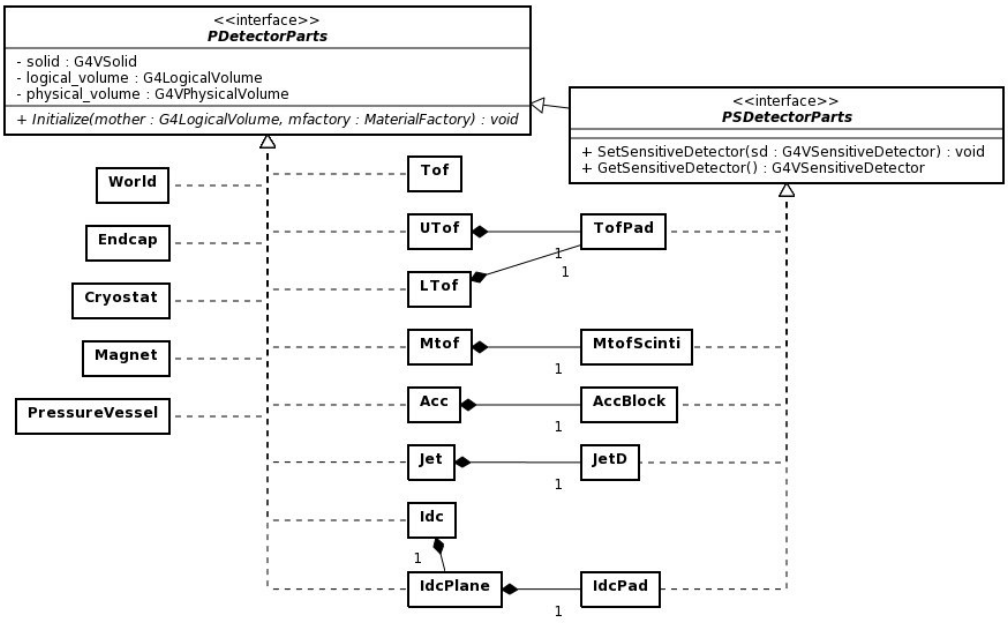

図 A.4: Detector Parts のクラス図

A.1.5 Copy Number Getter

Copy Number Getter のクラス図を図 A.5 に示す。

CopyNoGetterBase Copy Number Getter の基本クラス
各イベントのステップ情報からヒットした Detector の Copy Number を取得・生成する。GetCopyNumber 関数をオーバーライドし、Copy Number の設定を行う。PSensitiveDetector クラスに登録しておけば、Copy Number を取得に GetCopyNumber 関数が呼び出され、何も登録してないとヒットのあった Sensitive Detector の Copy Number がそのまま使用される。

IdcCopyNoGetter IDC 用 Copy Number Getter クラス
IDC は、親 Volume から順に IDC, IdcPlane, IdcPad という構造をしているので、それぞれの Copy Number を A, B, C とすると、$100A + 10B + C$ という Copy Number を生成する。

AccCopyNoGetter ACC 用 Copy Number Getter クラス
Sensitive Detector として登録する ACC Block の Copy Number は 0 しか設定されてないので、1 つ上のレベルの Volume は一意に決められているのでそちらを使用する。

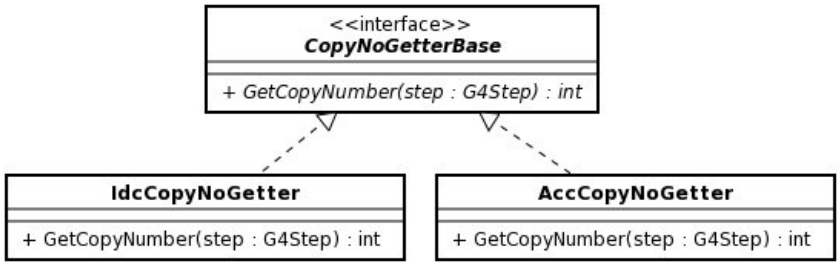

図 A.5: Copy Number Getter のクラス図

A.1.6 Wire Number Getter

Wire Number Getter のクラス図を図 A.6 に示す。

WireNumberGetterBase Wire Number Getter の基本クラス
JET/IDC の Wire Number を生成するためのクラス。PDcSensitiveDetector クラス登録する。Copy Number から Wire Number を取得する。派生クラスは GetWireNumber 関数をオーバーライドし、Copy Number から Wire Number への変換を実装する。

IdcWireNumberGetter IDC 用の Wire Number Getter クラス
IDC の Copy Number から Wire Number を再計算する。

JetWireNumberGetter JET 用の Wire Number Getter クラス
JET の Wire Number は配列で全て固定され決められている。

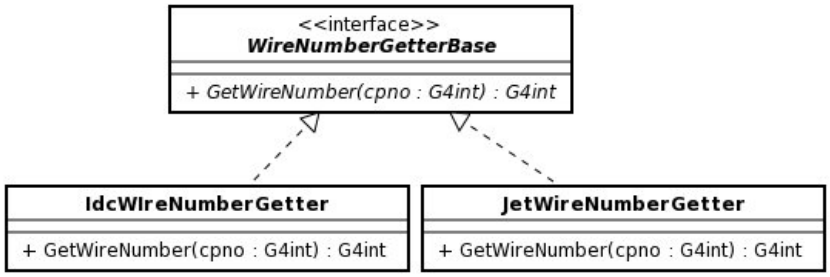

図 A.6: Wire Number Getter のクラス図

A.1.7 Magnetic Field

Magnetic Field のクラス図を図 A.7 に示す。

PMagneticField Magnetic Field 定義クラス
G4MagneticField を継承し、BESS の磁場を定義する。磁場の計算は p2lib の BessMagnet クラスに移譲する。PDetectorFactory の CreateMagnetField 関数でインスタンスを作成する。

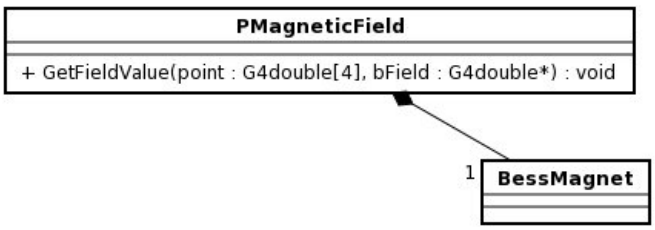

図 A.7: Magnetic Field のクラス図

A.2 Sensitive Detector

A.2.1 Sensitive Detector

Sensitive Detector のクラス図を図 A.8 に示す。Detector に Hit があるとこれらのクラスが呼び出されるのでヒット情報を Vector に格納する処理を行う。JET/IDC は PDcSensitiveDetector、ACC は PAccSensitiveDetector、UTOF, MTOF, LTOF は PTofSensitiveDetector を使う。

PSensitiveDetector Sensitive Detector の基本クラス

G4VSensitiveDetector クラスを継承しており、全ての Sensitive Detector はこのクラスを継承する。CopyNoGetterBase を持っているので Copy Number をカスタマイズする必要がある場合は登録しておく。CommonCut 関数で全 Sensitive Detector に共通のヒットのカットを定義する。この関数が false を返すとヒット情報は保存されない。Energy Deposit, Track Length の最小値を SetEdepLimit 関数、SetTrackLimit 関数で設定できる。この値はマクロからも設定できように、PSensitiveDetectorMessenger で実装されている。

PSensitiveDetectorMessenger Sensitive Detector 用マクロコマンド定義クラス

Sensitive Detector 用のマクロコマンドを定義する。以下のコマンドが定義されている（表 A.3)。引数の [] はデフォルト値である。コマンドディレクトリは「`/bess/sensitiveDetector/<Detector Name>`」である。`<Detector Name>` には Sensitive Detector 名が入る。

表 A.3: Sensitive Detector Commands

コマンド	引数 1	引数 2	内容
`setEdepLimit`	energy deposit[0]	単位 [MeV]	Energy Deposit の最小値を設定する
`getEdepLimit`			Energy Deposit の最小値を取得する
`setTrackLimit`	step length[0]		Step Length の最小値を設定する
`getTrackLimit`			Track Length の最小値を取得する

PDcSensitiveDetector JET/IDC 用 Sensitive Detector クラス

JET/IDC 用の Sensitive Detector クラス。PSensitiveDetector を継承している。Initialize 関数でマクロコマンドやヒットの定義を行う。ヒットがあれば ProcessHits 関数が呼び出されるので、ヒット情報を格納する処理を実装する。ヒットは PDcHitsCollection クラスに格納する。WireNumberGetterBase クラスを渡すと Wire Number の定義が変更できる。

PAccSensitiveDetector ACC 用 Sensitive Detector クラス

ACC 用の Sensitive Detector クラス。PSensitiveDetector を継承している。Initialize 関数でマクロコマンドやヒットの定義を行う。ヒットがあれば ProcessHits

関数が呼び出されるので、ヒット情報を格納する処理を実装する。ヒットは PAccHitsCollection クラスに格納する。

PTofSensitiveDetector TOF 用 Sensitive Detector クラス

TOF 用の Sensitive Detector クラス。PSensitiveDetector を継承している。Initialize関数でマクロコマンドやヒットの定義を行う。ヒットがあればProcessHits関数が呼び出されるので、ヒット情報を格納する処理を実装する。ヒットは PTofHitsCollection クラスに格納する。

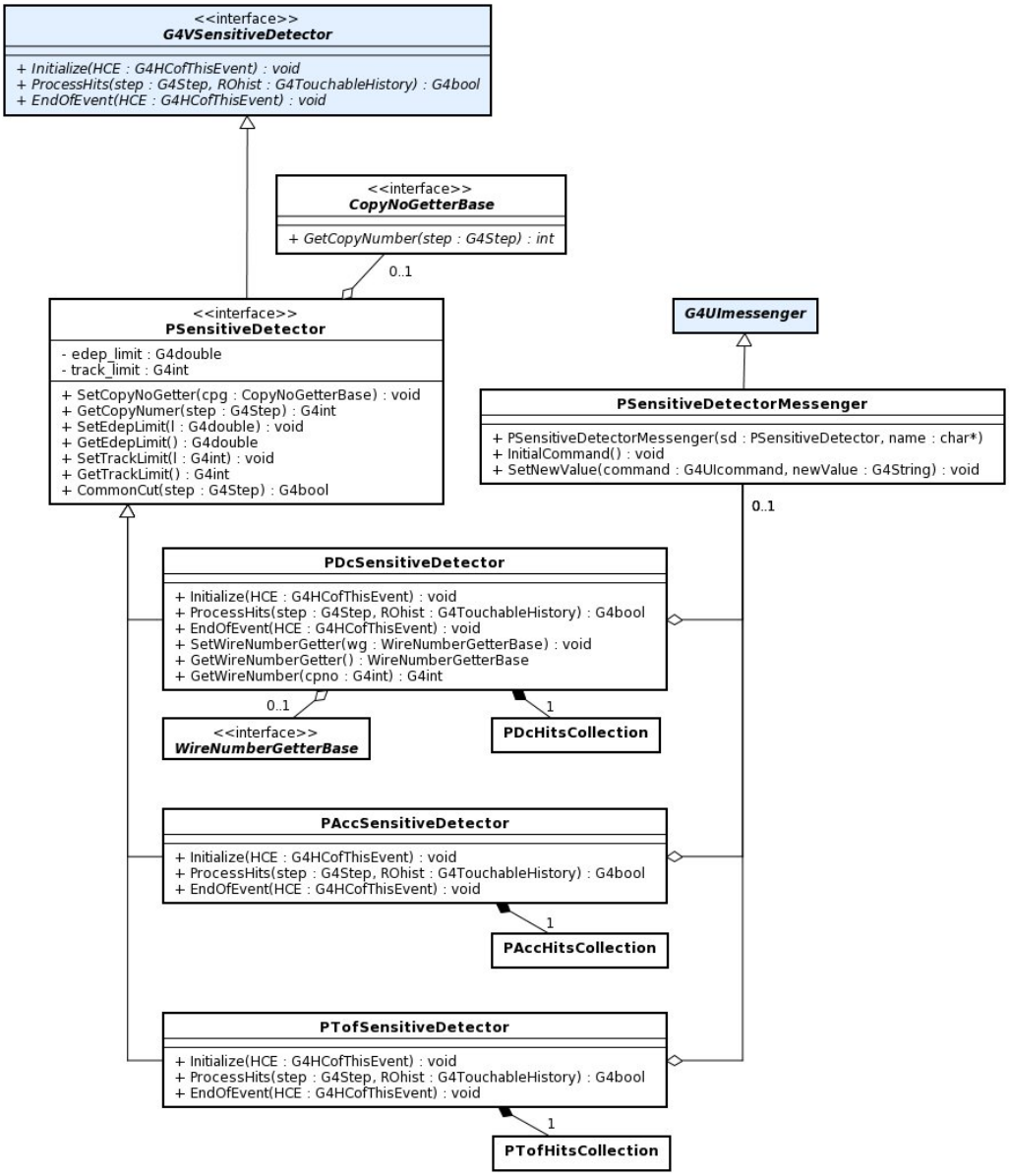

図 A.8: Sensitive Detector のクラス図

A.2.2 Hit Collection

Sensitive Detector のクラス図を図 A.8 に示す。P**Hit クラスが各 Detector の 1 Hit の情報を保持する。Hit が生成されたら、P**HitsCollection クラスに追加しておき、イベント終わりに保存する。各データの単位には GEANT4 のデフォルト単位を使用し、バイナリーファイルに出力する時に単位を指定する。

PTofHit TOF Hit データクラス

TOF の 1 Hit 分のデータを保持するクラス。表 A.4 のデータを保持する。生成した Hit 情報は PTofHitsCollection に登録する。

表 A.4: PTofHit のメンバ変数

変数名	型	説明
`copy_number`	G4float	Copy Number
`track_id`	G4int	Track ID
`hit_position`	G4ThreeVector	Hit Position (x,y,z)
`energy_deposit`	G4float	Energy Deposit
`step_length`	G4float	Step Length
`hit_time`	G4float	Hit Time

PDcHit JET/IDC Hit データクラス

JET/IDC の 1 Hit 分のデータを保持するクラス。表 A.5 のデータを保持する。生成した Hit 情報は PDcHitsCollection に登録する。

表 A.5: PDcHit のメンバ変数

変数名	型	説明
`copy_number`	G4float	Copy Number
`track_id`	G4int	Track ID
`hit_position`	G4ThreeVector	Hit Position (x,y,z)
`wire_number`	G4float	Wire Number
`energy_deposit`	G4float	Energy Deposit

PAccHit ACC Hit データクラス

ACC の 1 Hit 分のデータを保持するクラス。表 A.6 のデータを保持する。生成した Hit 情報は PAccHitsCollection に登録する。

PTofHitsCollection PTofHit の vector

G4THitsCollection<T> を PTofHit にバインドした Vector クラス。Run の前に PTofSensitiveDetector で G4SDManager に登録しておく。

表 A.6: PAccHit のメンバ変数

変数名	型	説明
`copy_number`	G4float	Copy Number
`track_id`	G4int	Track ID
`hit_position`	G4ThreeVector	Hit Position (x,y,z)
`charge`	G4float	Charge
`beta`	G4float	Beta
`energy_deposit`	G4float	Energy Deposit
`path_length`	G4float	Path Length

PDcHitsCollection PDcHit の vector

G4THitsCollection<T> を PDcHit にバインドした Vector クラス。Run の前に PDcSensitiveDetector で G4SDManager に登録しておく

PAccHitsCollection PAccHit の vector

G4THitsCollection<T> を PAccHit にバインドした Vector クラス。Run の前に PAccSensitiveDetector で G4SDManager に登録しておく

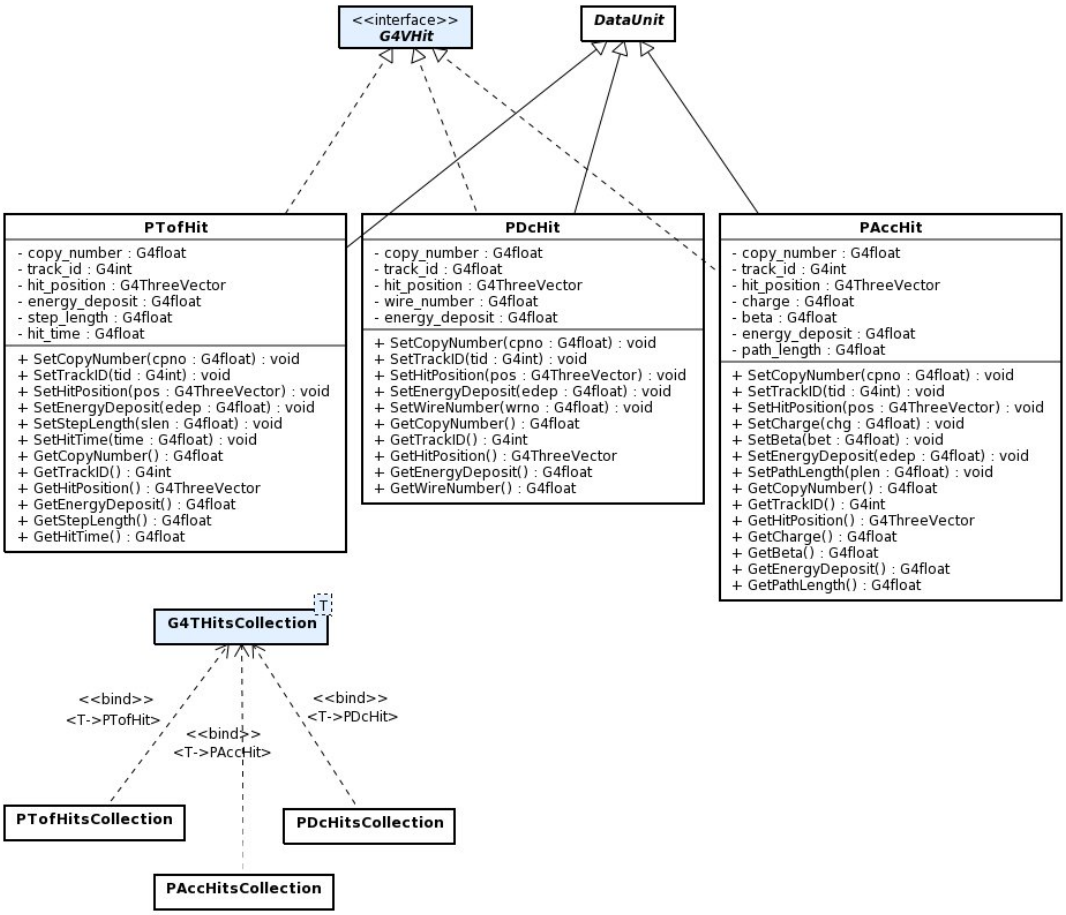

図 A.9: Hit Collection のクラス図

A.3 Action

A.3.1 Run Action

Run Action のクラス図を図 A.10 に示す。

PRunAction Run Action クラス

G4UserRunAction を継承している。BeginOfRunAction 関数、EndOfRunAction 関数をオーバーライドし、Run の始まりと終わり行う処理を実装する。

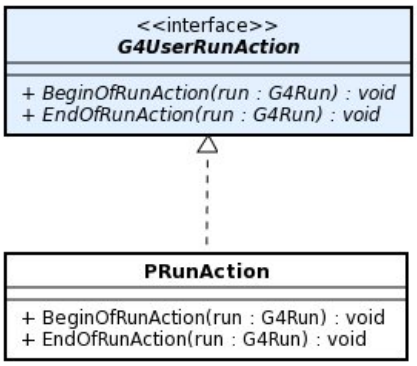

図 A.10: Run Actoin のクラス図

A.3.2 Event Action

Event Action のクラス図を図 A.11 に示す。

PEventAction Event Action クラス

G4UserEventAction を継承している。BeginOfEventAction 関数、EndOfEventAction 関数をオーバーライドしており、各イベントの最初と最後に行う処理を実装する。

A.3.3 Stepping Action

Stepping Action のクラス図を図 A.12 に示す。

PSteppingAction Step 毎の処理を定義するクラス

G4UserSteppingAction を継承し、オーバーライドした UserSteppingAction 関数にステップ毎の処理を実装する。ProcessTrack 関数は True Track 情報を監視・格納する。

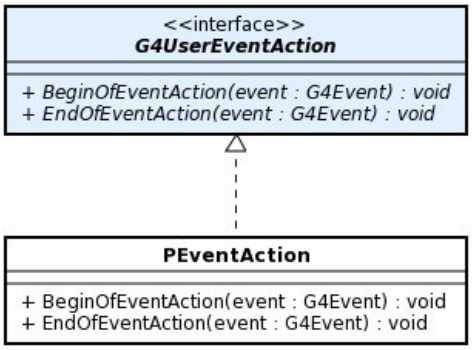

図 A.11: Event Actoin のクラス図

PEventGenerateSteppingAction Event Geterate 時の Step 毎の処理

G4UserSteppingAction を継承し、オーバーライドした UserSteppingAction 関数にステップ毎の処理を実装する。マグネット領域（$r < 53\text{cm}, |z| < 70\text{cm}$))) を通過したかをチェックし、EventGenerateTrigPat に設定する。

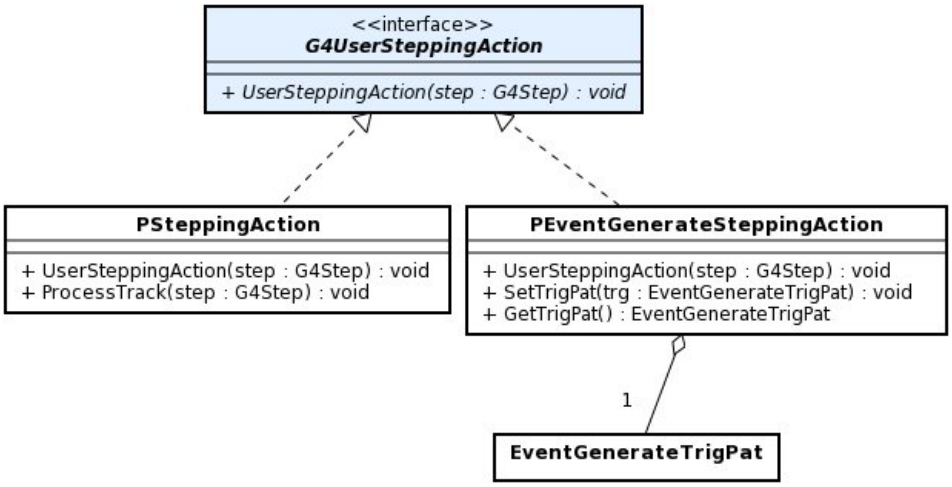

図 A.12: Stepping Actoin のクラス図

A.3.4 Generator Action

Generator Action のクラス図を図 A.13 に示す。毎 Event の始めに呼び出され、ビームの設定などを行う。

PEventGeneratorAction 乱数でイベント生成を行うクラス

G4VUserPrimaryGeneratorAction を継承したクラスで、GeneratePrimaries 関

数をオーバーライドして宇宙線の生成を実装する。一様乱数で半径WORLD_X(= 2 m) の球面上から入射する宇宙線（の初期位置、初期運動量、エネルギー）を生成する。Event Generate 時にはこのクラスを使用する。入射する粒子、エネルギー範囲は PEventGeneratorMessenger で実装されるマクロで設定可能。

PPrimaryGeneratorAction イベントデータからビームを設定するクラス
G4VUserPrimaryGeneratorAction を継承したクラスで、GeneratePrimaries 関数をオーバーライドして宇宙線の生成を実装する。PEventGeneratorAction で生成されたデータを読み込み、順に宇宙線として設定するクラス。イベントデータは BessEventPackage で保存されている。

PEventGeneratorMessenger イベント生成のマクロコマンドクラス
G4UImessenger クラスを継承し、PEventGeneratorAction 用のマクロコマンドを設定するクラス。このクラスでは以下のコマンドが設定される。コマンドディレクトリは「`/bess/eventGenerator/`」である。引数の [] はデフォルト値である。

表 A.7: Event Generator Command

コマンド	引数 1	引数 2	内容
`setParticle`	particle[proton]		入射粒子を設定する
`setEnergyMin`	energy[0.1]	単位 [GeV]	ランダム生成する粒子の エネルギーの最小値を設定する
`setEnergyMax`	energy[10]	単位 [GeV]	ランダム生成する粒子の エネルギーの最大値を設定する

A.3.5 Trigger Pattern

Trigger Pattern のクラス図を図 A.14 に示す。イベントを選択する時のトリガー条件を設定する。Event Generate と Simulation で条件が違うのでこれらのクラスで設定し、DataStorageManager に登録しておく。

TrigPatBase Trigger Pattern の基本クラス
Trigger Pattern の基本クラス。IsHit 関数でトリガー条件を満たすかをチェックして結果を返す。

BessTrigPat BESS 測定器の Trigger Pattern クラス
TrigPatBase を継承している。BESS のトリガー U+L, U+M を判定するクラス。

EventGenerateTrigPat Event Generate 時の Trigger Pattern クラス
TrigPatBase を継承している。Event Generate 時のトリガー、U+Mag を判定するクラス。Magnet 領域を通過したかは PEventGenerateSteppoingAction でチェックし、SetMagFlag 関数で設定しておく。

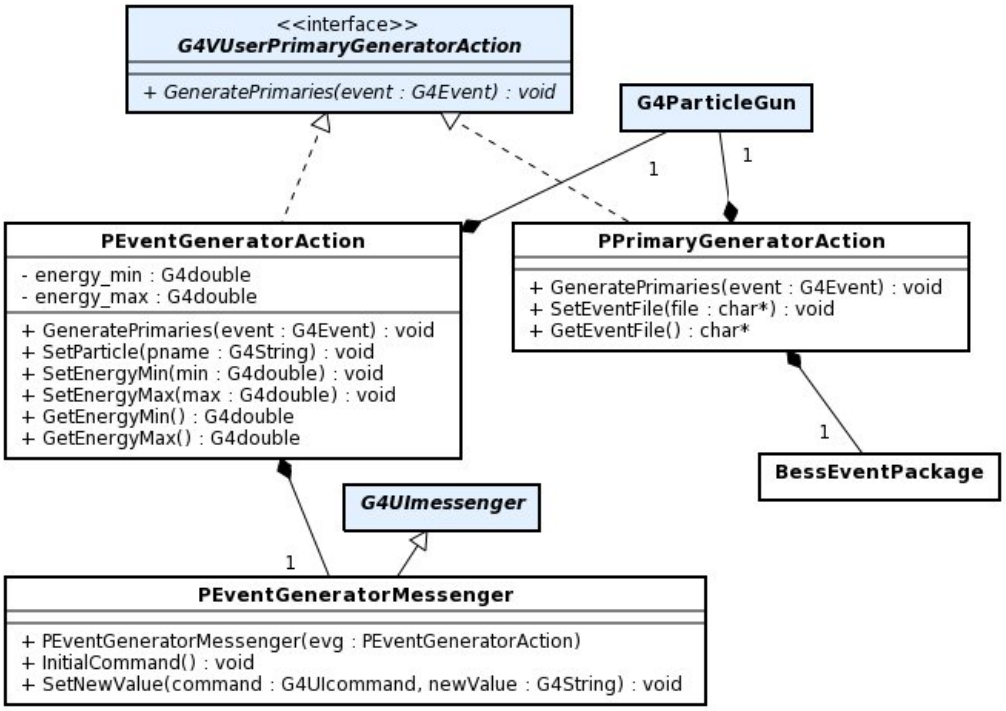

図 A.13: Generator Actoin のクラス図

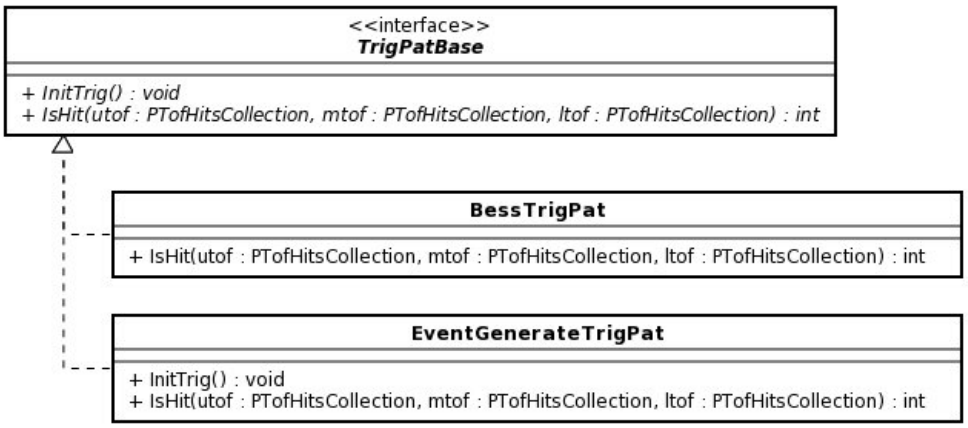

図 A.14: Trigger Pattern のクラス図

A.4 Physics

A.4.1 Physics List

Physics List のクラス図を図 A.15 に示す。

BessPhysicsList BESS の Physics List クラス
G4VModularPhysicsList クラスを継承している。GLAST の Physics(SSDecaPhysics, SSBosonPhysics, SSLeptionPhysics, SSNeutronPhysics, SShadronPhysics, SSIonPhysics) と BESS の Physics（BessPhysics）を登録する。

BessPhysics BESS の Physics クラス
G4VPhysicsConstructor を継承している。BESS の p, pbar の Cross Section を登録するクラス。陽子の Cross Section には、BessProtonElasticXS クラス, BessProtonInelasticXS クラスを登録する。反陽子の Cross Section には、BessAntiProtonElasticXS クラス、BessAntiProtonInelasticXS クラスを登録する。

(SSPhysics) GLAST の Physics List Package

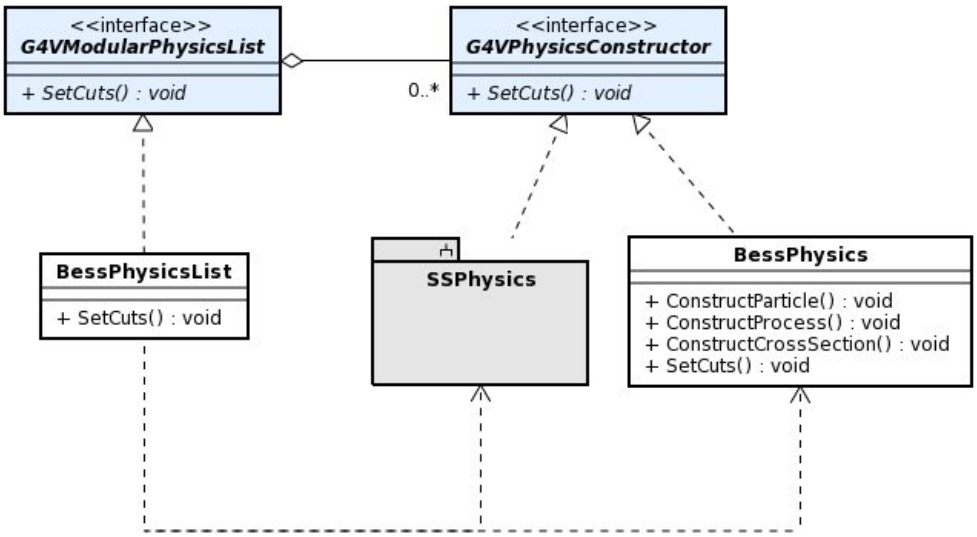

図 A.15: Physics List のクラス図

A.4.2 Proton Cross Section

陽子の Cross Section のクラス図を図 A.16 に示す。

BessProtonElasticXS BESS の Proton Elastic Cross Section

G4VCrossSectionDataSet クラスを継承している。BESS の Proton の Elastic Cross Section クラス。実装は BessProtonCrossSection に移譲し、GetCrossSection 関数は BessProtonCrossSection クラスの GetElasticCrossSection 関数を呼び出す。

BessProtonInelasticXS BESS の Proton Inelastic Cross Section

BESS の Proton の Inelastic Cross Section クラス。実装は BessProtonCrossSection に移譲し、GetCrossSection 関数は BessProtonCrossSection クラスの GetInelasticCrossSection 関数を呼び出す。

BessProtonCrossSections BESS の Proton Cross Section

BESS の Proton Cross Section を実装するクラス。GetElasticCrossSection 関数、GetInelasticCrossSection 関数で Cross Section を取得する。

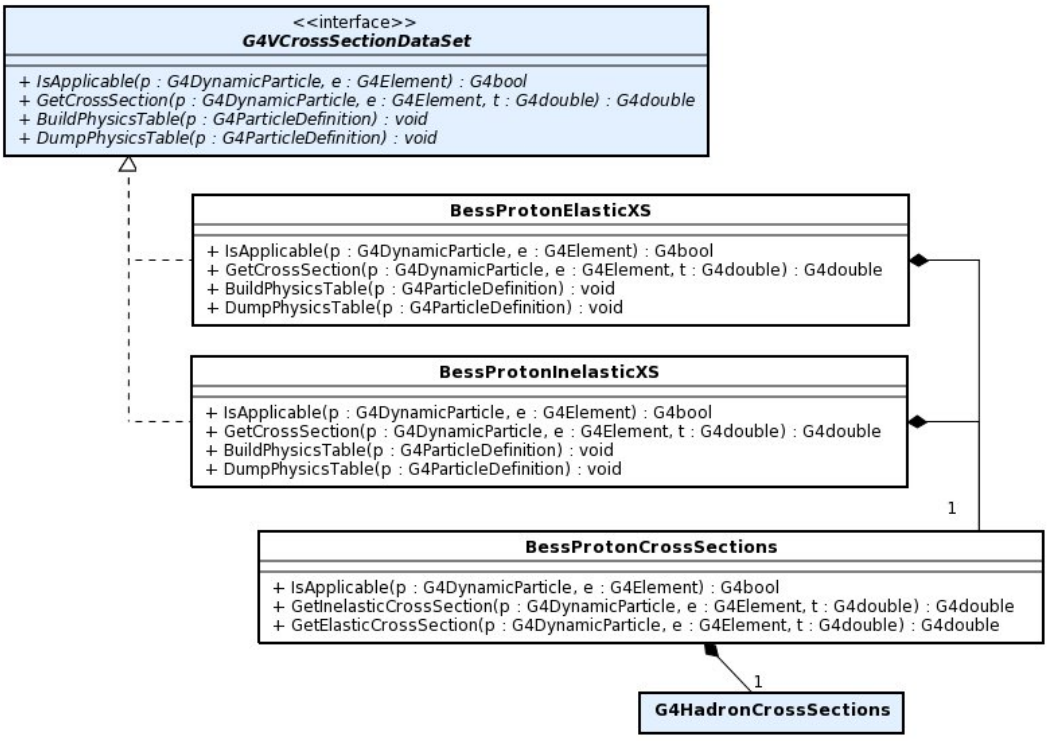

図 A.16: BESS の陽子の Cross Section のクラス図

A.4.3 Anti-proton Cross Section

反陽子の Cross Section のクラス図を図 A.17 に示す。

BessAntiProtonElasticXS BESS の Anti-proton Elastic Cross Section
G4VCrossSectionDataSet クラスを継承している。BESS の Anti-proton の Elastic Cross Section クラス。実装は BessAntiProtonCrossSection に移譲し、GetCrossSection 関数は BessAntiProtonCrossSections クラスの GetElasticCrossSection 関数を呼び出す。

BessAntiProtonInelasticXS BESS の Anti-proton Inelastic Cross Section
BESS の Anti-proton の Inelastic Cross Section クラス。実装は BessAntiProtonCrossSection に移譲し、GetCrossSection 関数は BessAntiProtonCrossSections クラスの GetInelasticCrossSection 関数を呼び出す。

BessAntiProtonCrossSections BESS の Anti-proton Cross Section
BESS の Anti-proton Cross Section を実装するクラス。GetElasticCrossSection 関数、GetInelasticCrossSection 関数で Cross Section を取得する。

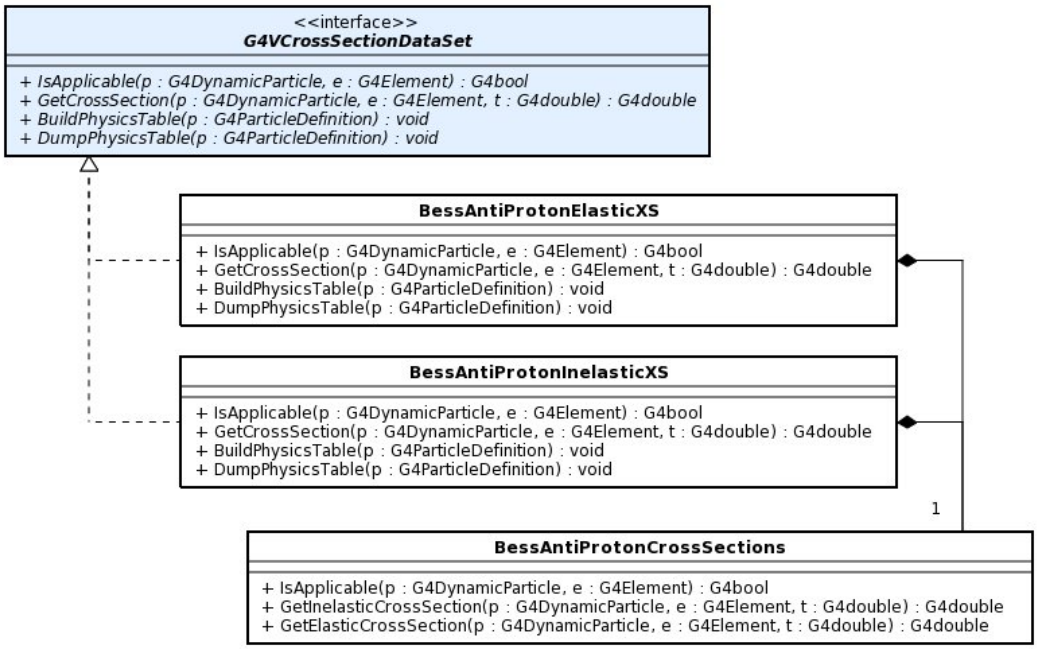

図 A.17: BESS の反陽子の Cross Section のクラス図

A.5 Data

A.5.1 Storage Manager

Storage Manager のクラス図を図 A.18 に示す。

DataStorageManager Data の Manager クラス

全てのヒットデータを管理するマネージャークラス。シングルトンで構成されているので GetDSManager 関数でインスタンスを取得する。AddStorage 関数で StorageTypeBase クラスを追加すれば、OpenStorage 関数で全ての Storage の Open 関数が呼び出され、StoreEvent 関数で Storage にデータが書き出される。Run が終われば CloseStorage 関数で全 Storage の Close 関数が呼び出される。イベントとして保存するかという Trigger 条件は TrigPatBase クラスとしてこの DataStorageManager に登録する。TrigPatBase が Hit とみなしたイベントのみ保存する。

DataStorageMessenger DataStorageManager 用マクロコマンドクラス

DataStorageManager 用マクロコマンド設定クラス。このクラスでは表 A.8 のコマンドが設定される。コマンドディレクトリは「`/bess/dataStorage/`」である。

表 A.8: Data Storage Manager Command

コマンド	引数 1	引数 2	説明
`setOutputAsRoot`	file name		出力する ROOT ファイル名を (.root) 設定する。
`setOutputAsBinay`	file name		出力する binary ファイル名 (.bin) を設定する。
`setOutputAsText`	file name		出力する text ファイル名 (.dat) を設定する

A.5.2 Data Storage

Data Storage のクラス図を図 A.19 に示す。StorageTypeBaes クラスを基本クラスとし、どういう形式で保存するかを各派生クラスで実装する。DataStorageManager からは Open 関数, Write 関数, Close 関数が呼び出される。

StorageTypeBase Storage の基本クラス

全ての Storage はこのクラスを基本クラスとし、Open 関数、Write 関数、Close 関数をオーバーライドする。

StorageByBinary Binary 出力の Stoarge

データを Binary 形式で HDD に保存する Storage クラス

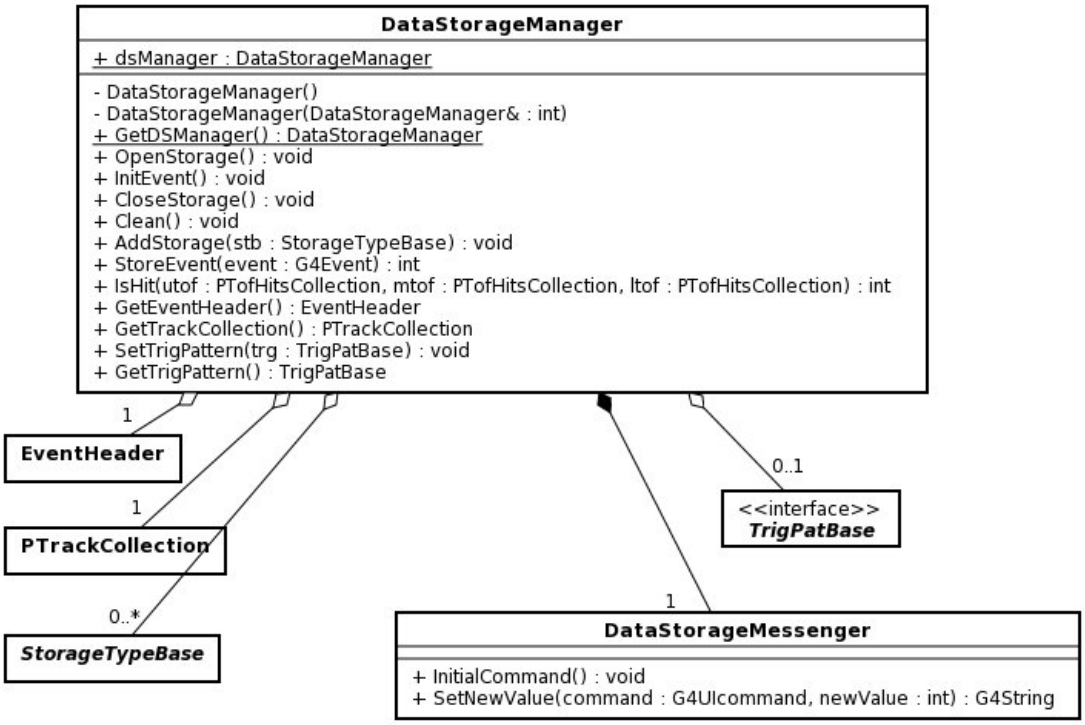

図 A.18: Storage Manager のクラス図

StorageByText Text 出力の Storage
データを Text 形式で HDD に保存する Storage クラス

StorageToRoot ROOT 出力の Storage
データを ROOT 形式で HDD に保存する Storage クラス

StorageToDisplay Display に出力
データを端末に表示する Storage クラス

GeneratedEventSelector Event Generate 用の Storage
Event Generate で生成したイベントデータを保存する Storage クラス。イベントデータは BessEventPackage 形式で出力する。

A.5.3 Data Format

Data Format のクラス図を図 A.20 に示す。

DatUnit データクラスの基本クラス
全てのデータクラスはこのクラスを継承する。

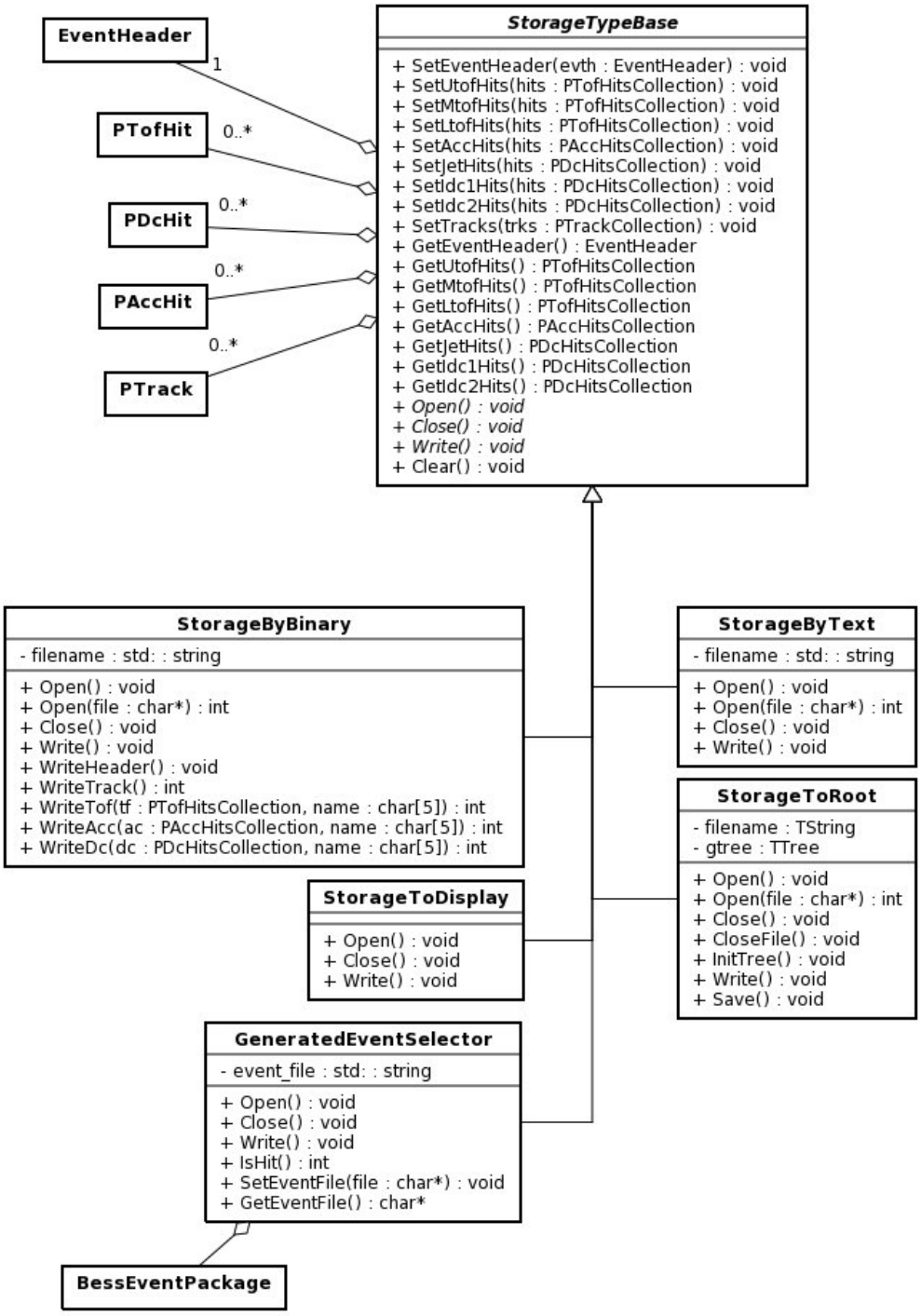

図 A.19: Data Storage のクラス図

表 A.9: PTrack のメンバ変数

変数名	型	説明
track_id	G4int	Track ID
parent_id	G4int	親 Track ID
particle_id	G4int	Particle の PDG Encoding
vertex	G4ThreeVector	Vertex (x,y,z)
momentum	G4ThreeVector	Momentum (p_x, p_y, p_z)
energy	G4float	Energy
track_time	G4float	Track Time (イベント開始時が 0)
n_step	G4int	Step 数

PTrack Track データクラス

1 Track 分のデータを保持するクラス。このクラスは表 A.9 のメンバ変数を所持する。

EventHeader Event Header クラス

1 Event の Header データを保持するクラス。このクラスは表 A.10 のメンバ変数を保持する。

表 A.10: EventHeader のメンバ変数

変数	型	説明
event_number	G4int	Event Number
event_id	G4int	Event ID
PDG	G4int	入射粒子の PDG Encoding
position	G4ThreeVector	入射粒子の Vertex (x,y,z)
momentum	G4Threevector	入射粒子の Momentum (p_x, p_y, p_z)
n_sd	G4int	ヒットのあった Sensitive Detector の種類
kinetic_energy	G4float	イベント開始時の入射粒子の Kinetic Energy

BessEvnetPackage Event Generate で生成するイベントデータクラス

Event Generate で生成したイベントデータを保持するクラス。このクラスは表 A.11 のメンバ変数を保持する。GEANT4 クラスとの結合を弱めるために、GEANT4 の型は使用していない。EVT_VEC は (x,y,z) の 3 つの変数を持つ構造体である。

PTrackCollection PTrack の vector std::vector≥T≤ を PTrack にバインドした vector クラスである。

表 A.11: BessEventPackage のメンバ変数

変数	型	説明
`pdg`	int	PDG Encoding
`event_number`	int	Event Number
`vertex`	EVT_VEC	Vertex (x,y,z)
`momentum`	EVT_VEC	Momentum (p_x, p_y, p_z)
`energy`	float	Energy

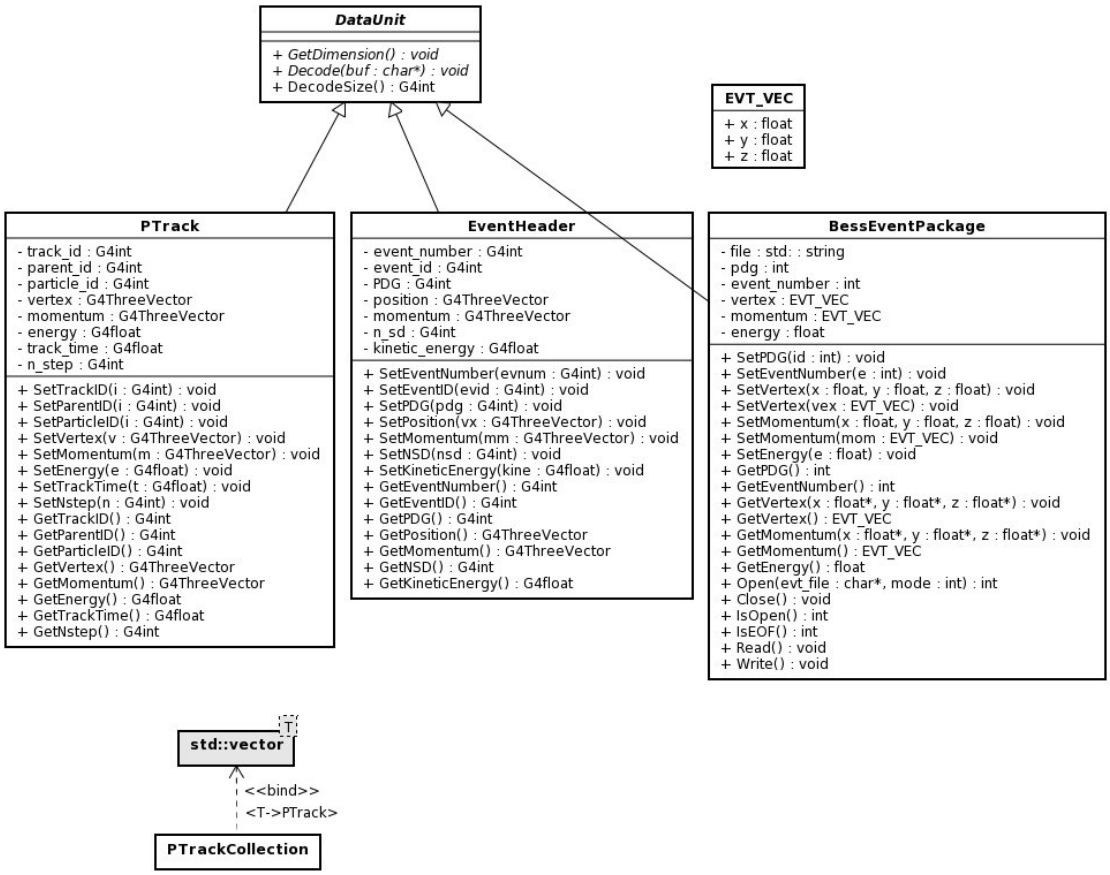

図 A.20: Data Format のクラス図

参考文献

[1] K.Yoshimura et al.(BESS collaboration). *Phys.Rev.*, Vol. 75, p. 3792, 1995.

[2] A.Yamamoto et al. *Adv. Space Res.*, Vol. 30, pp. 1253–1262, 2002.

[3] T.Yoshida et al. Bess-polar experiment. pp. 4027–4032, 2003.

[4] 松川陽介. Bess-polarii 実験 tof カウンター用 pmt 気密容器の開発及び低温低圧環境における pmt 動作試験. Master's thesis, 神戸大学, 2008.

[5] Haino Sadakazu. Measurement of the cosmic-ray low-energy antiproton spectrum with the first. 2009.

[6] 佐々木誠他. Bess-polarii フライト報告. 宇宙航空研究開発機構宇宙科学研究本部大気球シンポジウム, pp. 77–80, 2008.

[7] *kontron CP306*. http://emea.kontron.com/products/boards+and+mezzanines/3u+compactpci/x86+processor/cp306.html.

[8] *kontron CP307*. http://emea.kontron.com/products/boards+and+mezzanines/3u+compactpci/x86+processor/cp307.html.

[9] *SUGOI HUB*. http://www.system-talks.co.jp/product/sgc-40bhs.htm.

[10] J.D. Sullivan. *Nucl. Instrum. Methods*, Vol. 95, p. 5, 1971.

[11] Y. Asaoka et al. Measurement of low-energy antiproton detection efficiency in bess below 1 gev. *Nuclear Instruments and Methods in Physics Research A*, Vol. 489, pp. 170–177, 2002.

[12] Y. Asaoka et al. Absolute calibration of the antiproton detection efficiency for bess below 1 gev. *KEK Report*, Vol. 19, , 2001.

[13] R.K. Tripathi, F.A. Cucinotta, and J.W. Wilson. *Nucl. Instr. Meth. B*, Vol. 117, p. 347, 1996.