

(19) 日本国特許庁 (JP)

(12) 公表特許公報 (A)

(11) 特許出願公表番号

特表2013-527918

(P2013-527918A)

(43) 公表日 平成25年7月4日 (2013. 7. 4)

(51) Int. Cl.	F I	テーマコード (参考)
G O 1 T 1/20 (2006.01)	G O 1 T 1/20 B	2 G O 8 8
G O 1 T 1/00 (2006.01)	G O 1 T 1/00 A	
G O 1 T 1/12 (2006.01)	G O 1 T 1/12	
G O 1 T 1/203 (2006.01)	G O 1 T 1/20 E	
	G O 1 T 1/20 G	
審査請求 未請求 予備審査請求 未請求 (全 84 頁) 最終頁に続く		

(21) 出願番号 特願2013-502916 (P2013-502916)
 (86) (22) 出願日 平成23年4月4日 (2011. 4. 4)
 (85) 翻訳文提出日 平成24年11月28日 (2012. 11. 28)
 (86) 国際出願番号 PCT/US2011/031104
 (87) 国際公開番号 W02011/162851
 (87) 国際公開日 平成23年12月29日 (2011. 12. 29)
 (31) 優先権主張番号 61/320, 542
 (32) 優先日 平成22年4月2日 (2010. 4. 2)
 (33) 優先権主張国 米国 (US)

(71) 出願人 512254519
 ボード オブ トラスティーズ オブ ノーザン イリノイ ユニバーシティー
 アメリカ合衆国 イリノイ 60115,
 デカルブ, アダムズ ホール 321
 (71) 出願人 512255491
 フェルミ リサーチ アライアンス, エルエルシー
 アメリカ合衆国 イリノイ 60510-
 5011, バタビア, フェルミラボ,
 ビー. オー. ボックス 500
 (71) 出願人 512254885
 ユニバーシティー オブ ウロンゴン
 オーストラリア国 2522, ニュー
 サウス ウェールズ

最終頁に続く

(54) 【発明の名称】 3次元陽子コンピュータ断層撮影のための高性能計算 (HPC-PCT)

(57) 【要約】

撮像されるべき物体の第1の側に順番に並んだ2つの追跡検出器と、撮像されるべき物体の反対側に順番に並んだ2つの追跡検出器と、熱量計と、コンピュータクラスタを含み、追跡検出器がプラスチックシンチレーションファイバを含む、陽子コンピュータ断層撮影 (pCT) 検出器システム。検出器システム内の全ファイバは、シリコン光電子増倍管 (SiPM) によって読み出される。源から、2つの追跡検出器を通して、物体を通しておよび物体の周囲に、そして2つの反対側の追跡検出器を通して陽子を放出するステップと、熱量計によって、陽子のエネルギーを検出するステップと、物体を撮像するステップとによって物体を撮像する方法。

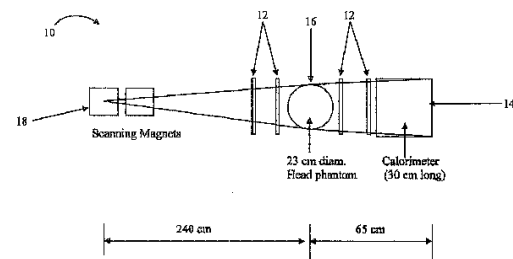


FIGURE 1

【特許請求の範囲】

【請求項 1】

陽子コンピュータ断層撮影（p C T）検出器システムであって、
該システムは、

撮像されるべき物体の第 1 の側に順番に並んだ 2 つの追跡検出器と、該撮像されるべき物体の反対側に順番に並んだ 2 つの追跡検出器と、熱量計と、コンピュータクラスタとを含み、該追跡検出器は、プラスチックシンチレーションファイバを含む、システム。

【請求項 2】

前記プラスチックシンチレーションファイバは、1 mm の直径を有する、請求項 1 に記載の p C T 検出器システム。

【請求項 3】

前記プラスチックシンチレーションファイバは、撮像場の全領域をカバーするように、1 列に充填される、請求項 1 に記載の p C T 検出器システム。

【請求項 4】

前記撮像場は、 27×36 cm である、請求項 3 に記載の p C T 検出器システム。

【請求項 5】

前記プラスチックシンチレーションファイバは、平面内に配列され、前記追跡検出器は、少なくとも 2 つの積層平面を含む、請求項 1 に記載の p C T 検出器システム。

【請求項 6】

前記 2 つの積層平面は、X および Y 方向に配向される、請求項 5 に記載の p C T 検出器システム。

【請求項 7】

前記 2 つの積層平面間に薄い発泡体状の材料をさらに含む、請求項 6 に記載の p C T 検出器システム。

【請求項 8】

前記プラスチックシンチレーションファイバは、正方形、円形、および六角形から成る群から選択される断面形状を有する、請求項 1 に記載の p C T 検出器システム。

【請求項 9】

2 つのプラスチックシンチレーションファイバが、各 S i P M に取着される、請求項 1 に記載の p C T 検出器システム。

【請求項 10】

1 つのプラスチックシンチレーションファイバが、各 S i P M に取着される、請求項 1 に記載の p C T 検出器システム。

【請求項 11】

前記追跡検出器は、前記ファイバおよび S i P M を支持するための支持手段をさらに含む、請求項 1 に記載の p C T 検出器システム。

【請求項 12】

前記プラスチックシンチレーションファイバは、PMMA（ポリ（メタクリル酸メチル））の外装材とともに、ポリスチレンから作製される、請求項 1 に記載の p C T 検出器システム。

【請求項 13】

前記熱量計は、波長シフティング（W L S）ファイバに取着されるシリコン光電子増倍管（S i P M）を含む、請求項 1 に記載の p C T 検出器システム。

【請求項 14】

前記熱量計は、シンチレータ板の積層を含み、各シンチレータ板は、1 つの S i P M と、1 つの W L S とを含む、請求項 13 に記載の p C T 検出器システム。

【請求項 15】

前記コンピュータクラスタは、前記追跡検出器および熱量計によって検出される前記陽子を分析するための分析手段をさらに含む、請求項 1 に記載の p C T 検出器システム。

【請求項 16】

10

20

30

40

50

前記分析手段は監督機コンピュータと、複数の作業機コンピュータとを含む、請求項 15 に記載の p C T 検出器システム。

【請求項 17】

前記物体を回転させるための回転ステージをさらに含み、前記追跡検出器は、封入体の内側にある、請求項 1 に記載の p C T 検出器システム。

【請求項 18】

物体を撮像する方法であって、

該方法は、

陽子を源から放出するステップであって、該陽子を 2 つの追跡検出器を通して、該物体を通しておよびその周りに、そして 2 つの反対側の追跡検出器を通して放出する、ステップと、

10

熱量計によって該陽子のエネルギーを検出するステップと、

該物体を撮像するステップと

を含む、方法。

【請求項 19】

前記放出するステップは、前記 2 つの追跡検出器および 2 つの反対側の追跡検出器において、前記陽子の位置を決定するステップをさらに含み、請求項 18 に記載の方法。

【請求項 20】

前記決定するステップは、前記追跡検出器の X および Y 平面において前記陽子を測定するステップをさらに含み、請求項 18 に記載の方法。

20

【請求項 21】

前記追跡検出器は、撮像場の全領域をカバーする 1 列に充填されたプラスチックシンチレーションファイバを含み、およびシリコン光電子増倍管を含む、請求項 18 に記載の方法。

【請求項 22】

前記熱量計は、波長シフティング (W L S) ファイバに取着されたシリコン光電子増倍管を含む、請求項 21 に記載の方法。

【請求項 23】

前記撮像ステップは、前記追跡検出器および熱量計から取得されたデータを、メッセージパッシングインターフェース (M P I) 規格を有する複数のコンピュータおよびグラフィック処理ユニットのクラスタ上で分析することを含む、請求項 18 に記載の方法。

30

【請求項 24】

前記撮像ステップは、コンパクトメモリ表現と、3D 画像空間全体を 1 度に解析することを含む、請求項 23 に記載の方法。

【請求項 25】

監督機コンピュータが、等量の陽子履歴を複数の作業機コンピュータおよび該監督機コンピュータに分配するステップと、積分電子密度 (I E D) および最尤パス (M L P) を計算するステップと、解ベクトルを解析してコンピュータ可読メモリ上に記憶するステップと、該解ベクトルのコピーを該監督機コンピュータに送信するステップと、該作業機コンピュータと該監督機コンピュータとの該解ベクトルを組み合わせ、該組み合わせられた解ベクトルをコンピュータ可読媒体上に記憶するステップと、該組み合わせられた解ベクトルを試験するステップと、該組み合わせられた解ベクトルが終了される場合に、前記物体の画像を生成するステップとをさらに含み、請求項 23 に記載の方法。

40

【請求項 26】

前記組み合わせられた解ベクトルが終了されない場合、該組み合わせられた解ベクトルを前記作業機コンピュータに伝送し、該組み合わせられた解ベクトルが終了されるまで、分配、計算、解析、送信、組み合わせ、試験ステップを繰り返し、そして前記物体の画像を生成する、請求項 25 に記載の方法。

【発明の詳細な説明】

【技術分野】

50

【0001】

(補助金情報)

本願の研究は、陸軍医療研究物資司令部の部門からの補助金によって一部支援を受けた(補助金第W81XWH-10-1-0170号)。政府は本発明に権利を有する。

【0002】

(技術分野)

本発明は、3次元計算に関する。特に、本発明は、陽子コンピュータ断層撮影に関する。

【背景技術】

【0003】

10

陽子コンピュータ断層撮影(pCT)による物体の内部3D画像の生成は、物体を通して多くの陽子を発射させることによって開始する。各陽子の経路(すなわち、トレース履歴)が記録され、全トレース履歴の集合が、画像を生成するコンピュータプログラムへの入力として使用される。pCTを通して生成された画像が、X線を通して生成された画像よりも癌の陽子ベースの治療において有効であるという示唆もある。

【0004】

陽子トレース履歴から画像を生成するアルゴリズムは、複雑かつ多くの段階を伴う。臨床的に有意義な時間枠内において、陽子ベースの癌治療を適用することができる(例えば、ヒトの頭部または骨盤)、物体の質の高い3D画像を生成するために必要とされるコンピュータメモリおよび速度は、今日利用可能な最も高性能なデスクトップコンピュータでさえ、その容量を上回る。したがって、陽子トレース履歴から画像をより効果的に作成するためのアルゴリズムの必要性が存在する。

20

【発明の概要】

【発明が解決しようとする課題】

【0005】

先行技術におけるpCTのための検出器は、より大型の嵩張るプラスチック製の立方体状の管(Pemlerら等)とともに、遥かに大容量を必要とする大型かつ嵩張る光子センサを採用しており、患者の周囲の空間が限定された陽子構台に搭載されるときに問題を呈する。Pemlerらは、放射線撮影を陽子によって行う可能性について説明しており、分析される物体の正面に、単一のX-Y平面を、その後も、単一X-Y平面を使用した。Pemlerらはまた、真空光電子増倍管、正方形ファイバ、および組み合わせ読み出し情報を使用する。Amaldiらは、30個の薄いプラスチックシンチレータの積層による10cm×10cmの撮像領域を有する陽子飛程放射線撮影システムについて説明している。Loma Lindaにおける現在のpCT検出器は、利用可能な領域(9cm×9cm)が限定されるシリコンウエハを使用しており、したがって、大領域(27×36cm)を達成するためには重複タイルの集積を必要とする。シリコンウエハから現在生成される最大サイズは10cm×10cmである。タイルは、屋根板状に重複される縁境界とともに設置され、検出器内のデッドスペース、または画像演算の際に「数学的な除去」を必要とする二重の層を生成する。

30

【0006】

40

したがって、また、デッドスペースおよびタイルの配列によって生成される二重の層を排除するpCTのための検出器に対する必要性が残留する。

【課題を解決するための手段】

【0007】

本発明は、撮像されるべき物体の第1の側に順番に並んだ2つの追跡(2D)検出器と、撮像される物体べきの反対側に順番に並んだ2つの追跡(2D)検出器と、熱量計とを含み、追跡検出器が、プラスチックシンチレーションファイバを含む陽子コンピュータ断層撮影(pCT)検出器システムを提供する。

【0008】

本発明はまた、陽子を、源から、2つの追跡検出器を通して、物体を通しておよび物体

50

の周囲に、そして2つの反対側の追跡検出器を通して放出するステップと、熱量計によって陽子のエネルギーを検出するステップと、物体を撮像するステップとによって物体を撮像する方法を提供する。

【0009】

本発明の他の利点は、付随の図面と関連して検討されるとき、以下の発明を実施するための形態を参照することによってより理解が深まり、容易に認識されるであろう。

【図面の簡単な説明】

【0010】

【図1】図1は、本発明のpCT検出器システムのレイアウトである。

【図2】図2は、本発明の中核概念である、pCT検出器システムの3次元表現である。

【図3A】図3A～3Cは、本発明の追跡検出器の一平面の3次元表現である。

【図3B】図3A～3Cは、本発明の追跡検出器の一平面の3次元表現である。

【図3C】図3A～3Cは、本発明の追跡検出器の一平面の3次元表現である。

【図4】図4は、本発明のシステムの3次元表現である。

【図5】図5は、スキャナシステムによって、画像を生成するプロセスを示す、流れ図である。

【図6】図6Aは、シリコンストリップ検出器の正面図である。図6Bは、検出された陽子ビームのグラフである。

【図7】図7は、追跡検出器のファイバの斜視図である。

【図8】図8は、熱量計の一部として、シンチレータ板の斜視図である。

【発明を実施するための形態】

【0011】

本発明は、陽子トレース履歴から3D画像を生成するための、概して、図1に示されるpCT検出器システム10を提供する。pCT検出器システム10は、好ましくは、4つの追跡検出器12と、熱量計14と、コンピュータクラスタ（図示せず）とを含む。

【0012】

追跡検出器12は、陽子のXおよびY平面における座標を検出するために使用される。各検出器12は、2つの平面、X平面およびY平面を有し、ファイバ（ストリップ）は、陽子トラックのXおよびY測定のために、適切な方向に配向される。各検出器は、その間に薄い発泡体状の材料を伴って、1つのXおよび1つのY平面を近接して設置することによって加工される。

【0013】

好ましくは、物体16の片側に2つ、物体16の反対側に2つと、撮像される物体16の周囲に4つの追跡検出器12が配列される。この配列は、さらに後述される。また、冗長性をもたらすために、物体16の各側に、3つ以上の追跡検出器12が存在することもできる。追跡検出器12は、1列に密接して充填され、撮像場の全領域、好ましくは、27cm×36cmまたは18cm×36cmをカバーする1mm直径の薄い円形プラスチックシンチレーションファイバ22を含む。追跡検出器12は、任意の他の好適なサイズであることができる。ファイバのコアは、好ましくは、PMMA（ポリ（メタクリル酸メチル））の外装材を有するポリスチレンである。追跡検出器12はまた、陽子の位置を検出するために、適切な電子機器を含み、このデータをコンピュータクラスタに中継する。

【0014】

陽子撮像のためのシンチレーションファイバの使用は、先行技術において報告されているが、本発明の追跡検出器12は、従来の検出器よりも高い空間解像度およびより大きな領域カバー率をもたらし、先行技術のタイル構造を排除する。最大6メートルを、死角を伴うことなく、これらのプラスチックシンチレーションファイバ22によって、カバーすることができる。より小さい円形ファイバ22は、先行技術デバイスにおいて使用される、より大きな正方形の管に比べて、ミリメートル未満レベルまでの優れた解像度をもたらす。本発明と先行技術との間の決定的な差異の1つは、SiPM（または、固体光電子増倍管）20を使用する一方、シンチレーションファイバ22を読み出すことである。従来

のデバイスは、光スペクトルの緑色領域における光の感度が不良である、嵩張る真空光電子増倍管の使用に基づいていた。先行技術は、光出力を増加させるために、厚い正方形シンチレーションファイバを使用していた。本発明の追跡検出器 12 は、約 2 倍高い光感度を有する SiPM20 を使用して、より薄いファイバ 22 の使用を可能にする。

【0015】

追跡検出器 12 は、図 1 および 2 に示されるように、2 つが、撮像されるべき物体 16 の第 1 の側に順番に並んで設置され、2 つが、物体 16 の反対側に順番に並んで設置されるように配列される。システム 10 の陽子ビームを放出する走査磁石（または、源）18 は、外側追跡検出器 12 のいずれかの側の熱量計 14 の反対側の位置に位置する。例えば、走査磁石 18 は、物体 16 の中心から 240 cm に位置することができ、熱量計 14 は、物体 16 の中心から 65 cm であることができる。内側追跡検出器 12 は、物体 16 の中心から 15 cm であることができる、外側追跡検出器 12 は、物体 16 の中心から 30 cm であることができる。他の距離も、必要に応じて使用することができる。

【0016】

追跡検出器 12 は、図 3A-3C に示されるように、先行技術に使用されるような光電管の代わりに、信号増幅のために、プラスチックシンチレーションファイバ 22 に装着される低コストのシリコン光電子増倍管 (SiPM) 20 の使用を含む。SiPM20 は、連続的カバー率を提供し、タイル構造の影響を排除する。SiPM は、光電管と対照的に、磁気によって影響を受けない。SiPM は、歴史的には、粒子検出器内で使用されてきた。SiPM は、本質的に、高速であって、事象間において約 100 ナノ秒の解析時間である。したがって、この設計は、従来の検出器システムよりも遥かに高いデータ取得率を可能にする。プラスチックシンチレーションファイバ 22 は、正方形（図 7 に示される）、円形、または六角形の断面形状を有することができる。SiPM20 を伴う追跡検出器 12 は、2 つのシンチレーションファイバ 22 を同時に含むことができ、代替として、1 つのファイバ 22 を 1 つの SiPM20 のために使用することができる。好ましくは、ファイバ 22 直径は、0.75 mm-1 mm の範囲内にある。ファイバ 22 は、陽子はその厚さを横断するとき光を生成するポリスチレンシンチレーションファイバであることができる。ファイバ 22 の 2 つの層は、1 つの層とは対照的に、検出効率を増加させる。ファイバ 22 の別の直交二重層は、空間内に陽子の 2D 座標を有するために、含まれることができる。図 3A-3B に示されるように、追跡検出器は、SiPM のための機械的支持部 24、ならびにシンチレーションファイバ 22 の設置のための支持としての役割を果たす、Rohacell 支持部 26 を含むことができる。追跡検出器 12 を生成するために、当技術分野において周知の任意の他の支持部を使用することができる。

【0017】

追跡検出器 12 の一実施例では、 $27 \times 36 \text{ cm}^2$ の領域（飛翔部分）が、カバーされる。シンチレーションファイバ 22 の選定された直径が 1 mm である場合、270 個のファイバ、270 個の SiPM、および 270 個の電子機器のチャンネルが必要とされる。これは、1 投影のみ (X) に対して該当する。2D 平面は、X および Y 投影 ($270 + 360 = 630$ チャンネル) を含む。総追跡検出器 12 は、約 $630 \times 4 = 2520$ チャンネルを含む。

【0018】

図 6A は、本発明の追跡検出器 12 の側面図を示す。この検出器 12 において、シリコンストリップの 2 つの層が存在する。SiPMs 20 に接続するシリコンセンサは、 $89.5 \text{ mm} \times 89.5 \text{ mm}$ であり、ストリップのピッチは、 $238 \mu\text{m}$ および $400 \mu\text{m}$ 厚である。各層のストリップは、6 つの ASIC*64 ストリップに個々に接続される。図 6B は、X および Y 座標を用いて追跡検出器 12 によって検出される陽子ビームの図を示す。

【0019】

熱量計 14 は、源 18 から放出される陽子エネルギーを検出するために使用される。直径 1.2 mm を有する SiPM20 は、1.2 mm 波長シフタ (WLS) ファイバ 36 を

10

20

30

40

50

通して、読み出し情報（デジタルまたはアナログ）を提供することができる。熱量計 14 内で使用される S i P M 20 は、追跡検出器 12 内で使用されるものと同一種類であることができる。熱量計 14 は、図 8 に示されるように、3 mm のシンチレータ板 34 の積層（少なくとも 2 つ）に配列することができる。この場合、チャンネルの総数は、約 120 である。これは、以下によって達することができる。1 つのシンチレータ板 34 は、1 つの W L S ファイバおよび 1 つの S i P M を含む。板の総数は、120 である。したがって、120 枚の板、120 個の W L S ファイバ、120 個の S I P M、および電子機器の 120 個のチャンネル（120 個のチャンネル）が存在する。熱量計 14 はさらに、適切な電子機器を含み、データをコンピュータクラスタに中継することができる。

【0020】

10

先行技術デバイス（A m a l d i、P r e m i e r）では、異なる組み合わせの読み出し情報システムが使用された。S i P M とシンチレータファイバおよび S i P M と W L S ファイバの組み合わせにおける使用は、以前はなかった。A m a l d i らは、熱量計読み出し情報のために S i P M および W L S ファイバを使用した。ガス光電子増倍管が、追跡検出器として使用された。P r e m i e r らは、シンチレーションファイバを読み出すための真空光電子増倍管と、熱量計を読み出すための真空光電子増倍管を使用した。記載された両方の場合、標的システム名は、放射線撮影である一方、本発明は、3D 断層撮影である。

【0021】

図 4 は、陽子暴露の間、物体 16 を保持および回転させる回転ステージ 28 を有するシステム 10 を示す。

20

【0022】

この検出器の一意的特徴は、再構築された画像内にアーチファクトを生成し得る死角または重複検出器を伴うことなく、大きい領域のカバー率を含むことである。この解決策は、シリコンストリップ検出器を使用する従来のアプローチよりも優れた信号対雑音比をもたらす。したがって、シリコン光電子増倍管に装着されたシンチレーションファイバは、背景雑音を減少させ、より鮮明な画像を制定し、撮像のための患者への線量を低減させる。これは、200 万／秒の速度で陽子を記録し、撮像時間を患者撮像に対して容認可能な時間である 10 分未満まで短縮する初の p C T 検出器である。

【0023】

30

本発明は、陽子を、源から 2 つの追跡検出器を通して、物体を通しておよび周囲に、そして 2 つの反対側の追跡検出器を通して放出するステップと、熱量計によって陽子を検出するステップと、物体を撮像するステップとによる物体を撮像する方法を提供する。

【0024】

より具体的には、陽子は、源 18（走査磁石）から、2 つの追跡検出器 12 を通して、撮像される物体 16 を通しておよびその周りに、そして付加的な 2 つの追跡検出器 12 を通して放出され、最後に、エネルギー検出器（すなわち、熱量計 14）（図 1）を通過する。十分なエネルギーの陽子は、人体を貫通することができ、追跡検出器 12 の入射および出射に応じて追跡することができる。陽子の X および Y 位置が測定され、追跡検出器 12 のそれぞれの X および Y 平面上で検出される。好ましくは、システム 10 は、4 つの検出器またはこれらのファイバ追跡器の 8 つの平面を含み、各平面を通して通過するのに伴って、各陽子トラックの位置を記録する。撮像される物体 16 は、2 つの検出器 12 が、画像の各側にあるように位置する。検出器 12 は、動作の間、物体の 360 度を周りを回転する。

40

【0025】

3D 陽子コンピュータ断層撮影画像の生成は、メモリ内に記憶される多数の陽子履歴からのデータを必要とする。先行技術において実装された独立型汎用目的グラフィカル処理ユニット（G P G P U）ワークステーション上で実行される従来のコンピュータプログラムは、コンピュータメモリ上の要求によって制約されていた。本明細書に説明されるアプローチは、再構築が、複数のワークステーションで同時に実行されるのと同様に、コンピ

50

ユータメモリに関する要求に限定されない。

【0026】

検出器および熱量計から記録された結果として生じたデータは、コンピュータクラスタ上の画像再構築ソフトウェアプログラムにフィードされ、高解像度画像を照射後10分以内に形成可能にすることができる。

【0027】

コンピュータプログラムは、メッセージパッシングインターフェース(MPI)規格に照らして書かれる。MPI規格に照らしてプログラムを書くことは、単一コンピュータとは対照的に、多くのコンピュータのクラスタ上でプログラムを起動することを可能にする。プログラムは、約768個のCPU(コンピュータ)に加えて、96個のGPU(グラフィック処理ユニット)上で起動されることができる。このように、多くのコンピュータの組み合わせられたメモリおよび演算能力を同時に担うことが可能であって、したがって、実質的に、プログラムを起動するためにかかる時間を(数桁も)短縮し、プログラムが撮像することができる課題サイズ(画像空間のサイズおよび画像の鮮明度によって測定される)を増加させる。したがって、熱量計14から取得されるデータは、この規格を有する複数のコンピュータのクラスタ上で分析することができる。

【0028】

また、コンピュータのメモリ内に画像を生成するために必要とされる情報を記憶するためのコンパクトな方法も開発された。このコンパクト設計は、問題を解決するために必要とされるメモリを数桁も削減する。独立型ワークステーション上で実行される、先行技術に実装された従来のコンピュータプログラムは、そのプログラムが必要とするコンピュータメモリの量によって制約されていた。

【0029】

対照的に、コンパクトメモリ表現は、コンピュータプログラムを配備し、3D画像空間全体を1度に解析することを可能にする。これは、先行技術において使用された「スライスおよびアセンブル」技法と比較して、より多くの陽子のトレース履歴からの情報を使用することを可能にするので、より高い品質の画像をもたらす。

【0030】

本発明の新しいコンピュータプログラムは、陽子療法によって治療される癌患者のための高品質3D内部画像を生成可能である。最も着目すべきは、新しいコンピュータプログラム、具体的には、MPIの追加およびコンパクトメモリ表現の2つの新規技法は、任意の他の周知の方法よりも高速で高品質な画像を生成することが可能であって、実際、そのように高速であることによって(すなわち、10分間未満)、プログラムは、今日の癌の陽子治療において使用されるX線生成画像を補強またはそれにとって代わるために使用することができるだけでなく、医療ケア提供者が、そのような画像に対して新しい用途を見出し、患者に送達されるケアの品質を改善する(例えば、陽子療法による患者の治療直前に、1セット以上の画像を生成し、治療計画を精緻化すること)ことができる。pCT撮像およびこの新しいコンピュータプログラムは、陽子療法によって治療される世界的癌患者すべてに対して、画像を生成するための最新技術となり、したがって、事実上の標準となり得る。

【0031】

コンピュータプログラムがさらに、以下の実施例1に説明されており、コードの実施例は、実施例2に示されている。概略すると、監督機コンピュータは、等量の陽子履歴を複数の作業機コンピュータおよび監督機コンピュータ自体に分配する。積分電子密度(IED)および最尤パス(MLP)が、コンピュータそれぞれによって計算される。解ベクトルが解析され、コンピュータのそれぞれの中のコンピュータ可読メモリ上に記憶される。作業機コンピュータからの解ベクトルのコピーが、監督機コンピュータに送信され、組み合わせられ、コンピュータ可読媒体上に記憶される。組み合わせられた解ベクトルは、監督機コンピュータによって試験され、組み合わせられた解ベクトルが終了したと決定される場合、画像が物体から生成される。組み合わせられた解ベクトルが、終了していないと決定され

10

20

30

40

50

る場合、組み合わされた解ベクトルは、作業機コンピュータに伝送され、前述のステップはそれぞれ、組み合わされた解ベクトルが、監督機コンピュータによって、終了されたと決定され、画像が、物体から生成されるまで繰り返される。

【0032】

図5は、本発明の一般のプロセスの流れ図を示す。陽子検出プロセスは、最終行に示され、追跡検出器12および熱量計14からのデータは、並列プロセッサおよび前述のアルゴリズムを使用して、上から分析される。最後に、概して、CT画像のような画像が生成される。

【0033】

したがって、このアプローチの利点は、画像を生成するために必要とされる時間およびメモリの両方を、それぞれ、数桁も大幅に削減することである。時間および空間の両方におけるこれらの劇的な削減は、画像が生成される方法を変化させるだけではなく、また、癌療法において使用することができる方法も変化させる。

【0034】

本発明はさらに、以下の実験的实施例を参照することによって、詳細に説明される。これらの実施例は、例示の目的のためだけに提供され、別様に規定がない限り、限定として意図されるものではない。したがって、本発明は、以下の実施例に限定されるものといかようにも解釈されるべきではなく、むしろ、本明細書に提供される教示の結果、明白となる、あらゆる変形例を包含するものと解釈されるべきである。

【0035】

(実施例1)

ブロック法のためのメモリおよびCPU数の評価

2010年3月

(1 アプローチ)

ストリング平均化法を実装するために我々が提案した解決策は、MPIとともに、一組の作業機プロセスを使用するものであって、1台の作業機が監督機として指定される。監督機は、陽子履歴を読み出し、全作業機にわたってそれを均一に分配し、それぞれのための均等な分担を保存する。履歴のその部分から、各作業機は、(1回だけ)IEDおよびMLPを計算し、ボクセル空間全体を表すその解ベクトルを初期化する(すなわち、各作業機は、解全体のうちのその独自のコピーを有する)。

【0036】

次いで、プログラムは、以下の反復ループに入る：

1. 各作業機は、そのIEDおよびMLPを使用して、その解のコピーを修正する。これは、その独自の停止基準を有する反復プロセスである；

2. 監督機ではない各作業機は、その解のコピーを監督機に送信する；

3. 監督機は、全他の作業機からの解ベクトルを収集し、独自のものと組み合わせ、組み合わされた解ベクトルを試験する。組み合わされた解が、「終了」である場合、監督機は、全他の作業機に終了を通知する。「終了ではない」場合、監督機は、組み合わされた解ベクトルを全ての他の作業機にブロードキャストし、次に、監督機とともに、次の反復のためのその開始点としてそれを使用する。

【0037】

(2 入力パラメータ)

撮像空間および計算リソースを特徴付ける以下の入力パラメータ

平面検出器パラメータ

P_h , P_w それぞれ、平面検出器の高さおよび幅(例えば、in cm)

P_r 検出器ストリップの解像度(例えば、238 μ)

撮像パラメータ

I_a 撮像陽子が発射される離散角度(360°)の数

ボクセル空間パラメータ

V_h , V_w , V_d それぞれ、ボクセル空間の高さ、幅、および奥行(例えば、in

10

20

30

40

50

c m)

V_r 画像の所望の解像度（例えば、1 mm）

V_s M L P 画像解像度オーバーサンプリング率（例えば、2）

V_p ボクセルあたりの陽子、すなわち、各ボクセルを進行する撮像陽子の数

計算リソースパラメータ

M プロセスあたりのメモリ

f 単精度浮動小数点を記憶するためのバイト数

p ポインタを記憶するためのバイト数

を定義する。

【0038】

（3 メモリ制約およびプロセス数）

本項では、プロセス M あたりのメモリによって制約される、 H_w で示す各作業機が処理することができる履歴の数を計算し、 H_w および履歴の総数から、問題を解決するために、 P によって示されるプロセスが必要とされる数を決定する。

【0039】

指数および計数。計数のために必要とされる最小数のバイトを計算し、また、種々の離散化空間を指数化するための反復的必要性が見出された。したがって、表記の便宜上、これらの用語を

・ V ; ボクセル空間内のボクセル数

・ b_v ; 線形化されたボクセル空間を指数化するために必要とされるバイト数

・ b_{ph} および b_{pw} ; それぞれ、寸法 P_h および P_w を指数化するために必要とされるバイト数

・ b_{Ia} ; 離散発射角度を指数化するために必要とされるバイト数

【0040】

【化1】

$$V = \left\lceil \frac{V_h V_w V_d}{V_r^3} \right\rceil \quad b_v = \left\lceil \frac{\lceil \log_2 V \rceil}{8} \right\rceil$$

$$b_{ph} = \left\lceil \frac{\lceil \log_2 \frac{P_h}{P_r} \rceil}{8} \right\rceil \quad b_{pw} = \left\lceil \frac{\lceil \log_2 \frac{P_w}{P_r} \rceil}{8} \right\rceil \quad b_{Ia} = \left\lceil \frac{\lceil \log_2 I_a \rceil}{8} \right\rceil$$

のように定義した。

【0041】

入力。各陽子履歴は、4つの $\langle x, y \rangle$ タプル（各検出器平面に対して1つ）、入力および出力エネルギー（ E_{in} および E_{out} ）、および投影角度から成る11-タプルによって特徴付けられる。これらの値は、4つの $\langle x, y \rangle$ タプルそれぞれに加えて、最後の3つの値に対する3つの浮動小数のために、 $b_{ph} + b_{pw}$ バイトのストレージを必要とする。各履歴は、ファントム（すなわち、空間カービング）の3-D表面を決定するように処理される。このプロセスでは、ボクセル空間は、各ボクセルに対して1つの要素と、アレイによって表され、アレイの要素は、ファントムが、それぞれ、ボクセルを占有するか、または占有しない場合、設定あるいはクリアされる。最初に、アレイへのアクセス時間を短縮するために、ビットではなく、バイトによって、各ボクセルを表すことを選択した。したがって、各作業機に対して処理された入力投影履歴を記憶するために必要とされる、メモリ（ I_m ）は、

【0042】

【数1】

$$I_m = H_w [4(b_{ph} + b_{pw}) + 3f] + V$$

$$= H_w \left[4 \left(\left\lceil \frac{\lceil \log_2 \frac{P_h}{P_r} \rceil}{8} \right\rceil + \left\lceil \frac{\lceil \log_2 \frac{P_w}{P_r} \rceil}{8} \right\rceil \right) + 3f \right] + \left\lceil \frac{V_h V_w V_d}{V_r^3} \right\rceil \quad (1)$$

となる。

【0043】

10

20

30

40

50

I E D。各陽子履歴は、浮動小数として記憶される積分電子密度（I E D）を計算するために使用される。各作業機に対してI E D（I E D_m）を記憶するために必要とされるメモリは、

【0044】

【数2】

$$IED_m = H_w f \quad (2)$$

である。

【0045】

M L P。各陽子履歴は、陽子が通過したそれらのボクセルを識別することによって、陽子の最尤パス（M L P）を計算するために使用される。略線形の軌道と仮定して、単一陽子の経路によって「接触される」ボクセル集合は、総ボクセル空間と比較して、非常に小さく、したがって、スパース表現が適切である。最初に、線形化されたボクセル空間内のその指数によって、ボクセルを識別することによって、陽子の経路内の各ボクセルをリスト化することを選択した（すなわち、各指数値に対して、b_v バイトを必要とする）。陽子履歴データは、いくつかの異なる角度のそれぞれから、多くの撮像陽子を発射することによって生成された。再び、略線形の軌道と仮定して、単一の陽子が通過するであろうボクセルの上限は、ボクセル空間立方体を通る対角線（l_d）の長さから合理的に概算することができる¹⁾（すなわち、

【0046】

【化2】

$$l_d = \sqrt{V_h^2 + V_w^2 + V_d^2}$$

）。陽子履歴は、所望の撮像解像度およびM L Pオーバーサンプリング率によって決定される解像度（すなわち、ステップサイズs）において評価される（すなわち、

【0047】

【化3】

$$s = \frac{V_z}{V_s}$$

）。陽子が通過するそれらのボクセルの識別に加え、最終画像の品質は、また、各ボクセルを通る陽子経路の弦長の評価を記録することによっても、改善することができることが示された（すなわち、「接触された」それらのボクセルを単に識別するのとは対照的である）。最初に、弦長、ボクセル立方体寸法（V_h、V_w、V_d）の関数としての浮動小数点数、および発射角度の評価を計算することを選択する。陽子履歴が、一組の作業機プロセスにわたってどのように分割されるかに関して、何ら仮定することなく（例えば、作業機は、単一発射角度、あるいは多くのまたは全部の発射角度から、全履歴を受信してもよい）、それを生成した弦長および角度をその角度から発射された陽子のボクセル指数に「接続」する単純メモリレイアウトが、以下のように、並行して、弦長のベクトルおよび2つの付加的ベクトルを維持する。

・弦長—これは、各離散発射角度に対して計算された弦長を記録する、浮動小数のベクトルである（すなわち、長さ＝I_a）。

・発射角度指数—これは、各陽子履歴が発射された角度を記録する、弦長の第1のベクトルに対する整数指数のベクトルである（すなわち、長さ＝H_w）。

・ボクセル指数アレイ—これは、各陽子履歴が、陽子はその経路上で「接触した」ボクセル指数のベクトル（評価長I_d）をポイントする、ポイントのベクトルである（すなわち、長さ＝H_w）。

【0048】

これらの概算および単純メモリレイアウト方式に基づいて、各作業機に対するM L Pボクセルおよびその弦長（M L P_m）を記憶するために必要とされるメモリの上限概算は、

10

20

30

40

50

【 0 0 4 9 】

【 数 3 】

$$\begin{aligned}
 MLP_m &= I_a f + H_w \{ b_{I_a} + p + (\lfloor \frac{l_d}{s} \rfloor + 1) b_v \} \\
 &= I_a f + H_w \{ (\lfloor \frac{\lceil \log_2 I_a \rceil}{8} \rfloor) + p + (\lfloor \frac{\sqrt{V_h^2 + V_w^2 + V_d^2}}{V_r} \rfloor + 1) (\lfloor \frac{\lceil \log_2 V \rceil}{8} \rfloor) \} \\
 &= I_a f + H_w \{ (\lfloor \frac{\lceil \log_2 I_a \rceil}{8} \rfloor) + p \\
 &\quad + (\lfloor \frac{V_s \sqrt{V_h^2 + V_w^2 + V_d^2}}{V_r} \rfloor + 1) (\lfloor \frac{\lceil \log_2 (\lfloor \frac{V_h V_w V_d}{V_r^3} \rfloor) \rceil}{8} \rfloor) \} \quad (3)
 \end{aligned}$$

10

となる。

【 0 0 5 0 】

（注記１）実際は、MLP評価は、入力の間カービング事前処理から生成された入射および出射の３－タプル＜x, y, z＞から開始ならびに終了するであろう。

【 0 0 5 1 】

解ベクトル。解ベクトルは、浮動小数のアレイであって、その長さは、ボクセルVの数によって決定される。各作業機が、解ベクトル全体のうちのその独自のコピーを維持することに着目し、したがって、各作業機において解ベクトル（V_m）を記憶するために必要とされるメモリは、

【 0 0 5 2 】

20

【 数 4 】

$$V_m = Vf = (\lfloor \frac{V_h V_w V_d}{V_r^3} \rfloor) f \quad (4)$$

となる。

【 0 0 5 3 】

作業機あたりの履歴。監督機を含む、各作業機は、前述のメモリ構成要素のそれぞれを記憶するために十分な空間を有していなければならないが、しかしながら、入力処理されると、そのメモは、解ベクトルのために再使用されてもよいことに着目する。

【 0 0 5 4 】

30

【 数 5 】

$$IED_m + MLP_m + MAX(I_m, V_m) \leq M \quad (5)$$

不等式（５）を書き換えると、そのメモリMによって制約される、各作業機が処理することができる履歴の最大数H_wは、

【 0 0 5 5 】

【 数 6 】

$$\begin{aligned}
 \underbrace{H_w f + I_a f + H_w \{ (\lfloor \frac{\lceil \log_2 I_a \rceil}{8} \rfloor) + p + (\lfloor \frac{2\sqrt{V_h^2 + V_w^2 + V_d^2}}{V_r} \rfloor + 1) (\lfloor \frac{\lceil \log_2 (\lfloor \frac{V_h V_w V_d}{V_r^3} \rfloor) \rceil}{8} \rfloor) \}}_{MLP_m} \\
 + MAX(\underbrace{H_w [4(\lfloor \frac{\lceil \log_2 \frac{P_h}{P_r} \rceil} \rfloor + \lfloor \frac{\lceil \log_2 \frac{P_w}{P_r} \rceil}{8} \rfloor) + 3f]}_{I_m}, \underbrace{(\lfloor \frac{V_h V_w V_d}{V_r^3} \rfloor, (\lfloor \frac{V_h V_w V_d}{V_r^3} \rfloor) f)}_{V_m}) \leq M \quad (6)
 \end{aligned}$$

40

のように計算することができる。

【 0 0 5 6 】

プロセスの数。ボクセルVの数およびボクセルV_pあたりの陽子の数によって、合計でいくつの入力陽子履歴が存在するかが分かる。これは、そのメモリによって制約される、各作業機が処理することができる履歴の最大数H_wと組み合わせて、

【 0 0 5 7 】

50

【数 7】

$$P = (\lceil \frac{V_p V}{H_w} \rceil) \quad (7)$$

Pで示すいくつかの作業機プロセスが、問題を解決するために必要であるかが分かる。

【0058】

(4 事例研究)

ここで、以下の値を使用して、 H_w および P を計算する。

Ph, Pw = 20 cm

Pr = . 238 mm

Ia = 180

Vh, Vw, Vd = 20 cm

Vr = 1 mm

Vs = 2

Vp = 10

M = 2 GB

f, p = 4 バイト

それらの値を得られた不等式 (6) に代入する²⁾。

【0059】

【数 8】

$$\begin{aligned} & \underbrace{4H_w + 180(4)}_{IED_m} + \underbrace{H_w \{ (\lceil \frac{\lceil \log_2 180 \rceil}{8} \rceil) + 4 + (\lfloor \frac{2\sqrt{3(20^2)}}{.1} \rfloor + 1) (\lceil \frac{\lceil \log_2 (\lceil \frac{20^3}{.1^3} \rceil) \rceil}{8} \rceil) \}}_{MLP_m} \\ & + \underbrace{MAX(H_w [4(\lceil \frac{\lceil \log_2 \frac{20}{.0238} \rceil} \rceil) + \lceil \frac{\lceil \log_2 \frac{20}{.0238} \rceil}{8} \rceil] + 3(4))}_{I_m} + \underbrace{[\frac{20^3}{.1^3}]}_{V_m}, (\lceil \frac{20^3}{.1^3} \rceil) 4 \\ & = \underbrace{4H_w + 720}_{IED_m} + \underbrace{H_w \{ 1 + 4 + (693)(4) \}}_{MLP_m} + \underbrace{MAX(28H_w + 8 \times 10^6)}_{I_m}, \underbrace{32 \times 10^6}_{V_m} \\ & = 2,781H_w + 720 + MAX(28H_w + 8 \times 10^6, 32 \times 10^6) \leq 2 GB \quad (8) \end{aligned}$$

事例 I $28H_w + 8 \times 10^6 \geq 32 \times 10^6$: 不等式 (8) の書換え

【0060】

【化 4】

$$\begin{aligned} 2,781H_w + 720 + 28H_w + 8 \times 10^6 & \leq 2 GB \\ H_w & \leq \frac{2 GB - 720 - 8 \times 10^6}{2,809} \\ H_w & \leq 761,652.9 \end{aligned}$$

事例 II $28H_w + 8 \times 10^6 < 32 \times 10^6$: 不等式 (8) の書換え

【0061】

【化 5】

$$\begin{aligned} 2,781H_w + 720 + 32 \times 10^6 & \leq 2 GB \\ H_w & \leq \frac{2 GB - 720 - 32 \times 10^6}{2,781} \\ H_w & \leq 760,691.5 \end{aligned}$$

事例 II の $V_m = 32 \times 10^6$ に基づいて、解を記憶するために必要とされるメモリは、入力 $I_m = 28H_w + 8 \times 10^6 = 29 \times 10^6$ を記憶するためのメモリの量より大きいため、事例 II に基づいて計算された値 $H_w = 760,692$ を使用する。その値を式 (7) に適用して、

【0062】

10

20

30

40

50

【化 6】

$$\begin{aligned}
 P &= \left\lceil \frac{V_p V}{H_w} \right\rceil \\
 &= \left\lceil \frac{80,000,000}{760,692} \right\rceil \\
 &= 106
 \end{aligned}$$

が必要であることが分かった

(注記 2) 整数で記憶することができるように、値 (

【0063】

【化 7】

$$\left\lceil \frac{\lceil \log_2(\lceil \frac{20^3}{1^3} \rceil) \rceil}{8} \right\rceil$$

) を 3 から 4 までに丸めた。

【0064】

(付録—スプレッドシート用)

不等式 (6) から、2 つの以下の事例に基づいて、 H_w を求める。

事例 I $I_m \geq V_M$ と仮定して、(6) における H_w を求める。

【0065】

【化 8】

$$\begin{aligned}
 \underbrace{\frac{H_w f}{IED_m} + I_a f + H_w \left\{ \left(\left\lceil \frac{\lceil \log_2 I_a \rceil}{8} \right\rceil \right) + p + \left(\left\lfloor \frac{V_s \sqrt{V_h^2 + V_w^2 + V_d^2}}{V_r} \right\rfloor + 1 \right) \left(\left\lceil \frac{\lceil \log_2(\lceil \frac{V_h V_w V_d}{V_r^3} \rceil) \rceil}{8} \right\rceil \right) \right\}}_{MLP_m} \\
 + \underbrace{H_w \left[4 \left(\left\lceil \frac{\lceil \log_2 \frac{P_h}{P_r} \rceil}{8} \right\rceil + \left\lceil \frac{\lceil \log_2 \frac{P_w}{P_r} \rceil}{8} \right\rceil + 3f \right) + \left\lceil \frac{V_h V_w V_d}{V_r^3} \right\rceil \right]}_{I_m} \leq M
 \end{aligned}$$

$$\begin{aligned}
 H_w [f + \{ \left(\left\lceil \frac{\lceil \log_2 I_a \rceil}{8} \right\rceil \right) + p + \left(\left\lfloor \frac{V_s \sqrt{V_h^2 + V_w^2 + V_d^2}}{V_r} \right\rfloor + 1 \right) \left(\left\lceil \frac{\lceil \log_2(\lceil \frac{V_h V_w V_d}{V_r^3} \rceil) \rceil}{8} \right\rceil \right) \} \\
 + [4 \left(\left\lceil \frac{\lceil \log_2 \frac{P_h}{P_r} \rceil}{8} \right\rceil + \left\lceil \frac{\lceil \log_2 \frac{P_w}{P_r} \rceil}{8} \right\rceil + 3f \right)] \leq M - I_a f - \left\lceil \frac{V_h V_w V_d}{V_r^3} \right\rceil
 \end{aligned}$$

$$\frac{M - I_a f - \left\lceil \frac{V_h V_w V_d}{V_r^3} \right\rceil}{4f + \left(\left\lceil \frac{\lceil \log_2 I_a \rceil}{8} \right\rceil \right) + p + \left(\left\lfloor \frac{V_s \sqrt{V_h^2 + V_w^2 + V_d^2}}{V_r} \right\rfloor + 1 \right) \left(\left\lceil \frac{\lceil \log_2(\lceil \frac{V_h V_w V_d}{V_r^3} \rceil) \rceil}{8} \right\rceil \right) + 4 \left(\left\lceil \frac{\lceil \log_2 \frac{P_h}{P_r} \rceil}{8} \right\rceil + \left\lceil \frac{\lceil \log_2 \frac{P_w}{P_r} \rceil}{8} \right\rceil \right)} > H_w$$

または、より便宜的表記を使用して表すと以下のようになる。

【0066】

【数 9】

$$\frac{M - I_a f - V}{4f + b_{Ia} + p + \left(\left\lfloor \frac{V_s(l_d)}{V_r} \right\rfloor + 1 \right) b_v + 4(b_{ph} + b_{pw})} > H_w \quad (9)$$

セクション 4 からのパラメータを代入して、 H_w を以下のように求める。

【0067】

10

20

30

40

【化 9】

$$\frac{2GB - (180)(4) - \lceil \frac{20^3}{.1^3} \rceil}{[4(4) + (\lceil \frac{\lceil \log_2 180 \rceil}{8} \rceil) + 4 + (\lfloor \frac{2\sqrt{3(20^2)}}{.1} \rfloor + 1)(\lceil \frac{\lceil \log_2(\lceil \frac{(20)^8}{.1^3} \rceil) \rceil}{8} \rceil) + 4(\lceil \frac{\lceil \log_2 \frac{20}{.0238} \rceil}{8} \rceil + \lceil \frac{\lceil \log_2 \frac{20}{.0238} \rceil}{8} \rceil)]} > H_w$$

$$\frac{2,139,482,928}{[16 + (1) + 4 + (692 + 1)(4) + 4(2 + 2)]} > H_w$$

$$761,652.9 > H_w$$

10

事例 I I $I_m < V_M$ と仮定して、(6)における H_w を求める。

【0068】

【化10】

$$\underbrace{\frac{H_w f + I_a f + H_w \{(\lceil \frac{\lceil \log_2 I_a \rceil}{8} \rceil) + p + (\lfloor \frac{V_s \sqrt{V_h^2 + V_w^2 + V_d^2}}{V_r} \rfloor + 1)(\lceil \frac{\lceil \log_2(\lceil \frac{V_h V_w V_d}{V_r^3} \rceil) \rceil}{8} \rceil)\}}_{MLP_m}}_{IED_m} + (\lceil \frac{V_h V_w V_d}{V_r^3} \rceil) f \leq M$$

20

$$H_w \{f + (\lceil \frac{\lceil \log_2 I_a \rceil}{8} \rceil) + p + (\lfloor \frac{V_s \sqrt{V_h^2 + V_w^2 + V_d^2}}{V_r} \rfloor + 1)(\lceil \frac{\lceil \log_2(\lceil \frac{V_h V_w V_d}{V_r^3} \rceil) \rceil}{8} \rceil)\} \leq M - I_a f - (\lceil \frac{V_h V_w V_d}{V_r^3} \rceil) f$$

$$\frac{M - \{I_a + (\lceil \frac{V_h V_w V_d}{V_r^3} \rceil)\} f}{f + (\lceil \frac{\lceil \log_2 I_a \rceil}{8} \rceil) + p + (\lfloor \frac{V_s \sqrt{V_h^2 + V_w^2 + V_d^2}}{V_r} \rfloor + 1)(\lceil \frac{\lceil \log_2(\lceil \frac{V_h V_w V_d}{V_r^3} \rceil) \rceil}{8} \rceil)} > H_w$$

30

または、より便宜的表記を使用して表すと、

【0069】

【数10】

$$\frac{M - (I_a + V)f}{f + b_{I_a} + p + (\lfloor \frac{V_s(l_d)}{V_r} \rfloor + 1)b_v} > H_w \quad (10)$$

となる。

【0070】

セクション4からのパラメータを(10)に代用して、再び、 H_w を以下のように求める³⁾。

【0071】

【化11】

$$\frac{2GB - \{180 + (\lceil \frac{20^3}{.1^3} \rceil)\}4}{4 + (\lceil \frac{\lceil \log_2 180 \rceil}{8} \rceil) + 4 + (\lfloor \frac{2\sqrt{3(20^2)}}{.1} \rfloor + 1)(\lceil \frac{\lceil \log_2(\lceil \frac{20^3}{.1^3} \rceil) \rceil}{8} \rceil)} > H_w$$

$$\frac{2,115,482,928}{4 + (1) + 4 + (693)(4)} > H_w$$

$$760,691.5 > H_w$$

50

(注記 3) 整数で記憶することができるように、値 (

【 0 0 7 2 】

【 化 1 2 】

$$\left\lceil \frac{\lceil \log_2(\lceil \frac{2n^2}{15} \rceil) \rceil}{8} \right\rceil$$

) を 3 から 4 までは丸めた。

【 0 0 7 3 】

【表 1】

```

EXAMPLE 2
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mpi.h>

const int RandomizedRange = 10000;

/* command line configurables */
int Randomized; /* -r <nelem> */
int N;

int parse_command_line_args(int argc, char **argv, int my_id)
{
    int i;
    int error;

    /* default values */
    Randomized = 0;
    N = 0;

    for (i = 1, error = 0; !error && i < argc; i++)
    {
        if (!strcmp(argv[i], "-r"))
        {
            Randomized = 1;
            if (i + 1 < argc && (N = atoi(argv[i+1])) > 0)
                i++;
            else
                error = 1;
        }
        else
            error = 1;
    } /* endfor */

    if (error && !my_id)
    {
        /* only Master prints usage message */
        fprintf(stderr, "\n\tusage: %s {-r <nelem>}\n\n", argv[0]);
        fprintf(stderr, "where\n\n");
        fprintf(stderr, "\t-r <nelem>\t- Randomized input with N=nelem\n\n");
    } /* endif */

    return error;
} /* end parse_command_line_args() */

struct stats
{
    float mean;
    float sse;
    unsigned int n;
}; /* end struct stats */

/*
"Note on a Method for Calculating Corrected Sums of Squares and Products,"
B.P. Welford, Technometrics, Vol 4, No. 3, Aug 1962, 419-420.

The Art of Computer Programming, D.E. Knuth, vol 2, 2nd Edition, p 216
The Art of Computer Programming, D.E. Knuth, vol 2, 3rd Edition, p 232

Reccurrence forumla that keeps a running mean (M_k) and running SSE (S_k)

```

【表 2】

```

for some stream of values X_i

For x_1
  M_1 = x_1
  S_1 = 0

For x_i i>1
  M_i = M_(i-1) + [(x_i - M_(i-1))/i]
  S_i = S_(i-1) + [(x_i - M_(i-1)) * (x_i - M_i)]

At any time k>1
  running mean = M_k
  running _sample_ (i.e., not _population_) variance = S_k/(k-1)

```

"Updating Formulae and a Pairwise Algorithm for Computer Sample Variances,"
T.F. Chan, G.H. Golub, and R.J. LeVeque,
Technical Report STAN-CS-79-773, Department of Computer Science,
Stanford University, 1979,
<ftp://reports.stanford.edu/pub/cstr/reports/cs/tr/79/773/CS-TR-79-773.pdf>

Given two sample sets where the mean and SSE have been computed
for each; $\langle M_{k,1} \ S_{k,1} \ N_1 \rangle$ and $\langle M_{k,2} \ S_{k,2} \ N_2 \rangle$
we can compute the combined mean and SSE as follows

```

N = N_1 + N_2
d = M_k,2 - M_k,1

combined mean = M_k,1 + d*N_2/N
combined SSE = S_k,1 + S_k,2 + d^2*N_1*N_2/N

*/

static void push(float *m, float *s, unsigned int *n, float x)
{
  *n = *n + 1;

  if (*n == 1)
  {
    *m = x;
    *s = 0.0;
  }
  else
  {
    float old_mean = *m;
    float d = x - old_mean;

    *m += (d/(*n));
    *s += (d*(x - *m));
  } /* endif */

  /* printf("after pushing %f RunningMean %f RunningSSE %f\n", x, *m, *s); */

  return;
} /* end push() */

void combine_stats(void *operand_vec,
                  void *operand_result_vec,
                  int *len,
                  MPI_Datatype *dtype)
{
  struct stats *op      = (struct stats *) operand_vec;
  struct stats *op_res = (struct stats *) operand_result_vec;

```

【表 3】

```

int i;
unsigned int combined_n;
float delta;

for (i = 0; i < *len; i++)
{
    combined_n = op[i].n + op_res[i].n;
    delta      = op[i].mean - op_res[i].mean;

    op_res[i].mean += (delta * op[i].n / combined_n);
    op_res[i].sse  += (op[i].sse +
                      (delta * delta * op[i].n * op_res[i].n / combined_n));
    op_res[i].n    = combined_n;
} /* endfor */

return;

} /* end combine_stats() */

#ifdef 0
void combine_stats(void *operand_vec,
                  void *operand_result_vec,
                  int *len,
                  MPI_Datatype *dtype)
{
    struct stats *op      = (struct stats *) operand_vec;
    struct stats *op_res = (struct stats *) operand_result_vec;
    int i;

    for (i = 0; i < *len; i++)
    {
        op_res[i].mean += op[i].mean;
        op_res[i].sse  += op[i].sse;
        op_res[i].n    += op[i].n;
    } /* endfor */

    return;

} /* end combine_stats() */
#endif

static float SampleData[] = {17.0, 19.0, 24.0};

int main(int argc, char **argv)
{
    int numprocs, my_id;
    int i;
    struct stats s[2];
    MPI_Op op;
    float *src_data;
    /* for derived struct */
    MPI_Datatype Stats_mpi;
    int block_lengths[3]; /* number of "things" in each element of */
                          /* our new struct. for us, 1 each */
    MPI_Aint displs[3];  /* displacement of each element */
    MPI_Datatype types[3]; /* type of each element */
    MPI_Aint start_addr;
    MPI_Aint addr;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_id);
    printf("hello, world: my_id %d numprocs %d\n", my_id, numprocs);

```

【表 4】

```

if (parse_command_line_args(argc, argv, my_id))
{
    MPI_Finalize();
    exit(1);
} /* endif */

if (Randomized)
{
    if ((src_data = malloc(N*sizeof(float))) == NULL)
    {
        fprintf(stderr, "ERROR: failed malloc %n bytes\n", N*sizeof(float));
        exit(1);
    } /* endif */

    for (i = 0; i < N; i++)
    {
        /* about 1/2 positive and 1/2 negative */
        src_data[i] = (random() % RandomizedRange)-(RandomizedRange/2);
        printf("my_id %d: sample_data %f\n", my_id, src_data[i]);
    }
}
else
{
    src_data = SampleData;
    N = sizeof(SampleData)/sizeof(float);
} /* endif */

/*****
/* compute local running mean and SSE */
*****/

s[0].n = s[1].n = 0;

for (i = 0; i < N; i++)
{
    /* just push the same sample data into both 'regions' of s[] */
    push(&(s[0].mean), &(s[0].sse), &(s[0].n), src_data[i]);
    push(&(s[1].mean), &(s[1].sse), &(s[1].n), src_data[i]);
} /* endfor */

/*****
/* compute global running mean and SSE */
*****/

block_lengths[0] = block_lengths[1] = block_lengths[2] = 1;

types[0] = MPI_FLOAT;
types[1] = MPI_FLOAT;
types[2] = MPI_UNSIGNED;

MPI_Address(&(s[0].mean), &start_addr);
displs[0] = 0;

MPI_Address(&(s[0].sse), &addr);
displs[1] = addr - start_addr;

MPI_Address(&(s[0].n), &addr);
displs[2] = addr - start_addr;

/* build new derived datatype */
MPI_Type_struct(3, /* nelements */
                block_lengths,
                displs,
                types,
                &Stats_mpi);

```

【表 5】

```

MPI_Type_commit(&Stats_mpi);

MPI_Op_create((MPI_User_function*) combine_stats, // my user-defined func
              1, // commute flag
              &op); // op handle

MPI_Allreduce(s, // operand
              s, // result
              2, // count
              Stats_mpi, // data type
              op, // operator
              MPI_COMM_WORLD);

MPI_Op_free(&op);

printf("myid %d: after s[0].mean %f s[0].sse %f s[0].n %d\n", my_id, s[0].mean, s[
0].sse, s[0].n);
printf("myid %d: after s[1].mean %f s[1].sse %f s[1].n %d\n", my_id, s[1].mean, s[
1].sse, s[1].n);

if (my_id == 0)
{
    /* computing mean and SSE the old-fashioned way for checksums */

    float sum = 0.0;
    float sse = 0.0;
    float mean;

    for (i = 0; i < N; i++)
        sum += (src_data[i] * numprocs);

    mean = sum/(N*numprocs);

    for (i = 0; i < N; i++)
        sse += ((src_data[i] - mean)*(src_data[i] - mean)*numprocs);
    printf("myid %d: 2-pass mean %f sse %f n %d\n",
           my_id, mean, sse, N*numprocs);

} /* endif */

if (Randomized)
{
    free(src_data);
} /* endif */

MPI_Finalize();

exit(0);

} /* end main() */
#ifdef PCT_IED_H
#define PCT_IED_H

#include "pct.h" // struct entry_exit

#if defined(__cplusplus)
extern "C" {
#endif

void compute_IED(int nhistories, float *input_floats, float *IED);

void adjust_IED(int nhistories,
                unsigned char * history_bitmask,
                unsigned short *input_ushorts,
                float *input_floats,

```

10

20

30

40

【表 6】

```

        float * IED,
        struct entry_exit_t *entry_exit);

#ifdef __cplusplus
}
#endif

#endif /* PCT_IED_H */
#ifndef __mlp_m__
#define __mlp_m__
10

/* here we use 'unsigned int's to store voxel indices */
class MLP_m
{
public:

    MLP_m(int my_nh);
    ~MLP_m();

    /**
    * proton histories */
    /**

void set_my_nhistories_for_MLP(int mnh_for_MLP)
    { my_nhistories_for_MLP = mnh_for_MLP; }
20

void add_voxel_idx_to_history(int h_idx, unsigned int v_idx);

/* iterators */
unsigned int *get_first_voxel_idx_from_history(int h_idx);
unsigned int *get_next_voxel_idx_from_history(int h_idx,
        unsigned int *last_voxel_idx_p);
/* faster iterator - assumes voxel idxs for a given history */
/* stored in a contiguous mem block */
unsigned int get_history_idx_to_ntouched_voxels_idx(int h_idx);

    /**
    * chord lengths */
    /**

float get_chord_length(int h_idx);
void register_firing_angle(int h_idx, float theta);
30

    /**
    * memory management stats */
    /**

int get_curr_mem_block_idx() { return curr_mem_block_idx; }
int get_n_mem_blocks() { return n_mem_blocks; }
unsigned long get_n_recorded_voxel_idx(int h_idx)
    { return n_recorded_voxel_idx; }
int get_my_nhistories_for_MLP() { return my_nhistories_for_MLP; }
unsigned long get_n_copies() { return n_copies; }
unsigned long get_n_copied_bytes() { return n_copied_bytes; }
unsigned long get_nbytes_allocd_for_voxel_idx(int h_idx);
int get_n_allocd_blocks() { return curr_mem_block_idx+1; }
40
unsigned int get_block_nbytes(int b_idx);

private:

    /**
    * proton histories */
    /**

/* my_nhistories = as many as I was given from input */

```

【表 7】

```

/* my_nhistories_for_MLP = number of histories remaining after */
/*                               throwing out as many as we know did */
/*                               not touch the phantom ... used only */
/*                               for initial big block malloc          */
int my_nhistories;
int my_nhistories_for_MLP;
unsigned long n_recorded_voxel_idx; // for estimating subsequent
                                   // big block malloc's

unsigned int **history_idx_to_touched_voxels_idx; // len=my_nhistories
unsigned int *history_idx_to_ntouched_voxels_idx; // len=my_nhistories

int n_mem_blocks;
unsigned int **mem_block_pointers; // len = n_mem_blocks
unsigned int *mem_block_nidx_lengths; // len = n_mem_blocks

/* slot for next voxel idx */
int curr_mem_block_idx;
unsigned int *next_voxel_idx_p;

/*****/
/* chord lengths */
/*****/

/* int n_firing_angles; */
float *chord_lengths; // len = N_FIRING_ANGLES
int *history_idx_to_chord_length_idx_map; // len = my_nhistories

void compute_chord_lengths();

/*****/
/* memory management stats */
/*****/
unsigned long n_copies;
unsigned long n_copied_bytes;

};

#endif
#ifndef PCT_PCT_H
#define PCT_PCT_H

#ifdef __MAIN__
#define EXTERN
#else
#define EXTERN extern
#endif

#include <iostream>
#include <cmath>
#include <math.h>
#include <sstream>
#include <vector>
#include <iomanip>
#include <fstream>
#include <string>

#include <cstdio>
#include <cstdlib>
#include <assert.h>

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

```

10

20

30

40

【表 8】

```

#include <unistd.h>

#include <time.h>
#include <sys/time.h>

#ifdef NO_MPI
#include <mpi.h>
#endif

#include "mlp_m.h"

using namespace std;

/*****
/* Problem Space */
*****/

/** START DEFINING THE SENSORS **/
// Data space definitions
// Sensor definitions - distances in CM, origin in center of volume
const float TRACK_Z1 = -30.0; // sensor plane closest to proton gun
const float TRACK_Z2 = -20.0; // 2nd sensor plane from gun
const float TRACK_Z3 = 20.0; // 3rd sensor plane from gun
const float TRACK_Z4 = 30.0; // 4th and last sensor plane from gun
const float TRACK_Z1_X = TRACK_Z1 + 0.1; // Slightly different distances for
const float TRACK_Z1_Y = TRACK_Z1 - 0.1; // x and y sensor planes
const float TRACK_Z2_X = TRACK_Z2 + 0.1;
const float TRACK_Z2_Y = TRACK_Z2 - 0.1;
const float TRACK_Z3_X = TRACK_Z3 - 0.1;
const float TRACK_Z3_Y = TRACK_Z3 + 0.1;
const float TRACK_Z4_X = TRACK_Z4 - 0.1;
const float TRACK_Z4_Y = TRACK_Z4 + 0.1;

// X-axis is HORIZONTAL as you stand behind the gun facing it shooting direction
// X-axis range values in CM
const float MIN_TRACK_X = -15.00;
const float MAX_TRACK_X = 15.00;
const int MAX_TRACK_X_IDX = 3000;

// Y-axis is VERTICAL as you stand behind the gun facing it shooting direction
// Y-axis range values in CM
const float MIN_TRACK_Y = -15.00;
const float MAX_TRACK_Y = 15.00;
const int MAX_TRACK_Y_IDX = 3000;

const float RANGE_TRACK_X = MAX_TRACK_X - MIN_TRACK_X;
const float RANGE_TRACK_Y = MAX_TRACK_Y - MIN_TRACK_Y;

// sensor resolution in CM
const float TRACK_DX = (RANGE_TRACK_X/MAX_TRACK_X_IDX);
const float TRACK_DY = (RANGE_TRACK_Y/MAX_TRACK_Y_IDX);

/** END DEFINING THE SENSORS **/

/** START DEFINING VOXEL SPACE BETWEEN SENSORS **/
/* The data reconstruction space has two coordinate systems. */

/* The first is measured in centimeters and has the same XYZ orientation */
/* as the sensor grid at beam rotation angle of 0. The origin is the at */
/* the center of the space between the sensor planes. X is positive to */
/* the right (as seen from the central origin). Y is positive up. Z is */
/* positive in the direction of the beam. */

/* The second coordinate system is based on a plane-major, row major ordering */

```

【表 9】

```

/* scheme with planes parallel to the beam rotation plane. The column axis */
/* is parallel to X. Row axis is parallel to Z. Plane axis is parallel to Y. */

/* The reconstruction volume is an array a[MAX_PLANE][MAX_ROW][MAX_COL] */

/* Minimum index values are always 0. Maximum values depend on the */
/* reconstruction bounds and the reconstruction resolution. */

// The desired bounds on the reconstruction volume may not be exactly
// what is used, due to sampling constraints.
// In code, use VOLUME_{MIN,MAX}_{X,Y,Z}. See below.
const float DESIRED_VOLUME_MIN_X = -14.00;
const float DESIRED_VOLUME_MAX_X = 14.00;
const float DESIRED_VOLUME_MIN_Z = -14.00;
const float DESIRED_VOLUME_MAX_Z = 14.00;
// The height
const float DESIRED_VOLUME_MIN_Y = -7.00;
const float DESIRED_VOLUME_MAX_Y = 10.00;

// Voxel sampling sizes in cm
const float D_COL = 0.1;
const float D_ROW = 0.1;
const float D_PLANE = 0.1;
const float D_DATA[3] = {D_COL, D_ROW, D_PLANE};

// NICK: these (later) need to be determined by specified distances and
// NICK: specified resolutions.
/* const int MAX_ROWS = 300; // Should be even */
/* const int MAX_COLS = 300; */
/* const int MAX_PLANES = 300; */
const int MAX_COLS =
    (int)((DESIRED_VOLUME_MAX_X - DESIRED_VOLUME_MIN_X) / D_COL);
const int MAX_ROWS =
    (int)((DESIRED_VOLUME_MAX_Z - DESIRED_VOLUME_MIN_Z) / D_ROW);
const int MAX_PLANES =
    (int)((DESIRED_VOLUME_MAX_Y - DESIRED_VOLUME_MIN_Y) / D_PLANE);

const int MAX_DATA[3] = {MAX_COLS, MAX_ROWS, MAX_PLANES};

const float VOLUME_MIN_X = DESIRED_VOLUME_MIN_X;
const float VOLUME_MAX_X = VOLUME_MIN_X + MAX_COLS * D_COL;
const float VOLUME_MIN_Y = DESIRED_VOLUME_MIN_Y;
const float VOLUME_MAX_Y = VOLUME_MIN_Y + MAX_PLANES * D_PLANE;
const float VOLUME_MIN_Z = DESIRED_VOLUME_MIN_Z;
const float VOLUME_MAX_Z = VOLUME_MIN_Z + MAX_ROWS * D_ROW;

// NICK: VOXEL_CUBE_DIAG_VOXEL_LEN and NVOXELS (later) need to be corrected
// NICK: when MAX_{ROWS,COLS,PLANES} are converted from nvoxels (which is
// NICK: what they are now) to cm ... VOXEL_CUBE_DIAG_VOXEL_LEN and
// NICK: must both be measured in nvoxels
const float VOXEL_CUBE_DIAG_VOXEL_LEN = sqrt((MAX_ROWS*MAX_ROWS)
    + (MAX_COLS*MAX_COLS)
    + (MAX_PLANES*MAX_PLANES));

const int NVOXELS = MAX_COLS*MAX_ROWS*MAX_PLANES;
/**/ END DEFINING VOXEL SPACE BETWEEN SENSORS ***/

// for space carving and entry-exit, cut-off between "hit" or "miss" phantom
const float ENERGY_THRESHOLD = 190.0;
const float ENERGY_DIFF_THRESHOLD = 2.7;

const float ELECTRON_DENSITY_WATER = 1.0;
const float ELECTRON_DENSITY_AIR = 0.001;
const float INITIAL_SOLUTION_INSIDE_PHANTOM = ELECTRON_DENSITY_WATER;
const float INITIAL_SOLUTION_OUTSIDE_PHANTOM = ELECTRON_DENSITY_AIR;

```

【表 10】

```

// for MLP
// constants in nature
const float RADIATION_LENGTH_WATER = 36.1; //cm
// not used const float RADIATION_LENGTH_AIR = 28818; //units?
// distance in CM between planes on one side of phantom
const float SSD_SEPARATION = (TRACK_Z2-TRACK_Z1);
// distance in CM between innermost planes
const float SEPARATION = (TRACK_Z3-TRACK_Z2);

// no longer used const float DIMENSION = 256;
// something in CM
// no longer used const float RECON_SIZE = 21;
// no longer used const float PIXEL_SIZE = RECON_SIZE/DIMENSION;
// in cm
const float STEP_LENGTH = .05; //originally -> 0.5 * PIXEL_SIZE;
const int NUM_STEPS = (int) (SEPARATION/STEP_LENGTH);

const int N_FIRING_ANGLES = 180; // should be factor or multiple of 360
const float THETA_IDX_PER_DEGREE = ((float) N_FIRING_ANGLES)/360.0);
inline int degrees_theta_to_idx(float theta)
{ return ((int) (theta*THETA_IDX_PER_DEGREE + 0.5)); }
inline float idx_to_degrees_theta(int idx)
{ return ((float) idx)/THETA_IDX_PER_DEGREE; }

/*****
/* For Linear Solvers */
*****/

const float DEFAULT_RELAXATION = 0.3;

/*****
/* For Memory Management */
*****/

const int MLP_MEMBLOCK_PTRS_PER_ALLOC = 4;
// const float PORTION_HISTORIES_FOR_MLP = 0.95; // estimate portion
// of histories remaining
// after throwing out
// histories we know
// don't touch phantom

const float PORTION_HISTORIES_FOR_MLP = 0.60;
/* const float PORTION_VOXEL_CUBE_DIAG_FOR_MLP = 0.60; */
/* const float PORTION_VOXEL_CUBE_DIAG_FOR_MLP = 0.40; */
const float PORTION_VOXEL_CUBE_DIAG_FOR_MLP = 0.30;
/* for 38M, setting PORTION_VOXEL_CUBE_DIAG_FOR_MLP = 0.20, */
/* results in initial alloc .2GB when we need .3GB */
/* const float PORTION_VOXEL_CUBE_DIAG_FOR_MLP = 0.20; */

EXTERN char Nowstring[100];
EXTERN char Relaxparamstring[100];

/* mem mgmt counters */
EXTERN unsigned long N_mallocs;
EXTERN unsigned long N_reallocs;
EXTERN unsigned long N_allocated_bytes;
EXTERN unsigned long Largest_single_mem_request;
EXTERN char Largest_single_mem_request_fname[500];
EXTERN int Largest_single_mem_request_line_no;
#ifdef DEBUG
EXTERN unsigned long N_frees;
EXTERN unsigned long N_freed_bytes;
EXTERN unsigned long N_allocated_bytes_high_water_mark;
EXTERN unsigned long N_outstanding_allocated_bytes;
struct alloc_entry
{

```

【表 1 1】

```

void *p;
unsigned long len;
};
const int N_ALLOC_ENTRIES = 1000;
EXTERN struct alloc_entry Allocs[N_ALLOC_ENTRIES];
EXTERN int Next_alloc_idx;
#endif

/* theoretically possible (but very unlikely) that the first alloc'd
/* block is not big enough to hold all the voxel idxs touched by the
/* the first history alone.
/* in that case we need to REALLOC the first block (not malloc another
/* because all the voxel idx's from a single history must reside in a
/* single allocated block (i.e., they cannot span multiple blocks).
/* we could change that 'no span' design decision if we want to, but
/* that will at least require us to change the MLP::get_{first,next}
/* iterators used by the linear solvers_and_ (worst yet) add another
/* vector len=my_nhistories of idxs that would index mem_block_pointers[]
/* to record the 'starting' block for the iterators.
/* we also use this value to pad the estimate for remaining block allocs.
/* ... in short, set this value relatively low (i.e., ~10%).
/* const float PERCENT_PAD_SUBSEQUENT_BLOCK_ALLOCS = 0.10; */
const float PERCENT_PAD_SUBSEQUENT_BLOCK_ALLOCS = 0.20;

/* computed from previous settings */
const int AVERAGE_TOUCHED_VOXELS_PER_HISTORY_EST =
    (int) (PORTION_VOXEL_CUBE_DIAG_FOR_MLP*VOXEL_CUBE_DIAG_VOXEL_LEN);

/*****
/* Other Useful Constants */
/*****
const int DEFAULT_MAXHISTORIES_PER_WORKER = 1000000;
const int UNSPECIFIED_NINPUT_HISTORIES = -1;
const int UNSPECIFIED_MAX_NINPUT_HISTORIES_PER_FILE = -1;

/*****
/* MPI Stuff */
/*****

const int ForemanRank = 0;
EXTERN int Myid, Nprocs;

/* MPI_Abort codes */
const int ABORT_INVALID_CMD_LINE_ARGS = 1;
const int ABORT_FAILED_MALLOC = 2;
const int ABORT_FAILED_FOPEN = 3;
const int ABORT_TOO_MANY_HISTORIES = 4;
const int ABORT_INSUFFICIENT_INPUT = 5;
const int ABORT_INVALID_IDX = 6;
const int ABORT_INVALID_PTR = 7;
const int ABORT_INVALID_ARG = 8;
const int ABORT_TRACE_OUTSIDE_VOLUME = 9;
const int ABORT_INCONSISTENT_INTERNAL_STATE = 10;
const int ABORT_STATS_ERROR = 11;

/* MPI Message tags */
const int INPUT_FLOAT_TAG = 1;
const int INPUT_USHORT_TAG = 2;

/*****
/* Input Data Layout */
/*****

/* Describing input_ushorts */

```

【表 1 2】

```

const int X1_DISPL = 0;
const int Y1_DISPL = 1;
const int X2_DISPL = 2;
const int Y2_DISPL = 3;
const int X3_DISPL = 4;
const int Y3_DISPL = 5;
const int X4_DISPL = 6;
const int Y4_DISPL = 7;
const int INPUT_USHORT_STRIDE = 8;
/* Describing input_floats */
const int EIN_DISPL = 0;
const int EOUT_DISPL = 1;
const int THETA_DISPL = 2;
const int INPUT_FLOAT_STRIDE = 3;

/*****
/* Utility Functions and Macros */
*****/

// A 3-D array in C/C++ defined a[NPLANES][NROWS][NCOLS] is stored in memory
// with changes in the NPLANES dimension having the largest stride
// and changes in the NCOLS dimension with the smallest stride.
inline int convert_3D_voxel_to_idx(int p_idx, int r_idx, int c_idx)
{
    return ((p_idx * MAX_ROWS * MAX_COLS)
            + (r_idx * MAX_COLS)
            + c_idx);
} /* end convert_3D_voxel_to_idx() */

inline void convert_sensor_to_voxel(float dst[3],
                                   float src[3],
                                   float cos_theta,
                                   float sin_theta)
{
    // Input is in cm.
    // Convert to world coordinates
    // Assuming clockwise rotation about y axis as seen from positive y
    // Correct assumption?
    /*
    dst[0] = src[0] * cos_theta - src[2] * sin_theta;
    dst[1] = src[0] * sin_theta + src[2] * cos_theta;
    dst[2] = src[1];
    */

    // No. Rotation is counterclockwise about y axis as seen from positive y
    dst[0] = src[0] * cos_theta + src[2] * sin_theta;
    dst[1] = -src[0] * sin_theta + src[2] * cos_theta;
    dst[2] = src[1];

    // Translate to data origin and scale to data space.
    dst[0] = (dst[0] - VOLUME_MIN_X) / D_COL;
    dst[1] = (dst[1] - VOLUME_MIN_Z) / D_ROW;
    dst[2] = (dst[2] - VOLUME_MIN_Y) / D_PLANE;

    return;

} // end convert_sensor_to_voxel()

/* input_ushort accessors */
inline unsigned short get_x1(unsigned short *v, int idx)
{ return v[(idx*INPUT_USHORT_STRIDE) + X1_DISPL]; }
inline unsigned short get_y1(unsigned short *v, int idx)
{ return v[(idx*INPUT_USHORT_STRIDE) + Y1_DISPL]; }
inline unsigned short get_x2(unsigned short *v, int idx)

```

【表 1 3】

```

    { return v[(idx*INPUT_USHORT_STRIDE) + X2_DISPL]; }
    inline unsigned short get_y2(unsigned short *v, int idx)
    { return v[(idx*INPUT_USHORT_STRIDE) + Y2_DISPL]; }
    inline unsigned short get_x3(unsigned short *v, int idx)
    { return v[(idx*INPUT_USHORT_STRIDE) + X3_DISPL]; }
    inline unsigned short get_y3(unsigned short *v, int idx)
    { return v[(idx*INPUT_USHORT_STRIDE) + Y3_DISPL]; }
    inline unsigned short get_x4(unsigned short *v, int idx)
    { return v[(idx*INPUT_USHORT_STRIDE) + X4_DISPL]; }
    inline unsigned short get_y4(unsigned short *v, int idx)
    { return v[(idx*INPUT_USHORT_STRIDE) + Y4_DISPL]; }
    /* input_floats accessors */
    inline float get_ein(float *v, int idx)
    { return v[(idx*INPUT_FLOAT_STRIDE) + EIN_DISPL]; }
    inline float get_eout(float *v, int idx)
    { return v[(idx*INPUT_FLOAT_STRIDE) + EOUT_DISPL]; }
    inline float get_theta(float *v, int idx)
    { return v[(idx*INPUT_FLOAT_STRIDE) + THETA_DISPL]; }

    /* Abort stuff */
    #ifdef NO_MPI
    #define PCT_ABORT(CODE) assert(0);
    #else
    #define PCT_ABORT(CODE) MPI_Abort(MPI_COMM_WORLD, (CODE));
    #endif

    /* Bitmask stuff */
    #define CLEAR(v, b) (v)[(b)/8] &= (~(0x1 << (7-((b)%8))) & (0xff))
    #define SET(v, b) (v)[(b)/8] |= (0x1 << (7-((b)%8)))
    #define TEST(v, b) ((v)[(b)/8] & (0x1 << (7-((b)%8))))

    /* Memory Management */

    /* these two I want kept a #define so that __LINE__ and __FILE__ resolve */
    /* to the place in the source code where the potential error occurred */
    #ifdef NO_MPI
    #ifdef DEBUG
    /* NO_MPI DEBUG */
    /* must use assert(0) instead of MPI_Abort() */
    #define PCT_MALLOC(P, S, T)
    {
        if ((S) <= 0)
        {
            fprintf(stderr,
                "ERROR: %s %d: proc %d: "
                "malloc called for %ld bytes\n",
                __FILE__, __LINE__, Myid,
                (long) (S));
            assert(0);
        }
        if (!(P) = (T *) malloc((S)))
        {
            fprintf(stderr,
                "ERROR: %s %d: proc %d: "
                "failed malloc %ld bytes\n",
                __FILE__, __LINE__, Myid,
                (long) (S));
            assert(0);
        }
        N_mallocs++;
        N_allocated_bytes += (S);
        if (Next_alloc_idx == N_ALLOC_ENTRIES)
        {
            fprintf(stderr,

```

10

20

30

40

【表 1 4】

```

        "ERROR: %s %d: proc %d: " \
        "Next_alloc_idx %d (max %d)\n", \
        FILE_, LINE_, Myid, \
        Next_alloc_idx, \
        N_ALLOC_ENTRIES-1); \
    assert(0); \
} \
Allocs[Next_alloc_idx].p = (P); \
Allocs[Next_alloc_idx].len = (S); \
Next_alloc_idx++; \
N_outstanding_allocated_bytes += (S); \
if (N_outstanding_allocated_bytes \
    > N_allocated_bytes_high_water_mark) \
{ \
    N_allocated_bytes_high_water_mark = \
        N_outstanding_allocated_bytes; \
} \
if ((unsigned long) (S) \
    > Largest_single_mem_request) \
{ \
    Largest_single_mem_request = (S); \
    strcpy(Largest_single_mem_request_fname, FILE_); \
    Largest_single_mem_request_line_no = LINE_; \
} \
} \
#else \
/* NO_MPI !DEBUG */ \
#define PCT_MALLOC(P, S, T) \
{ \
    if (!((P) = (T *) malloc((S)))) \
    { \
        fprintf(stderr, \
            "ERROR: %s %d: proc %d: " \
            "failed malloc %ld bytes\n", \
            FILE_, LINE_, Myid, \
            (long) (S)); \
        assert(0); \
    } \
    N_mallocs++; \
    N_allocated_bytes += (S); \
    if ((unsigned long) (S) \
        > Largest_single_mem_request) \
    { \
        Largest_single_mem_request = (S); \
        strcpy(Largest_single_mem_request_fname, FILE_); \
        Largest_single_mem_request_line_no = LINE_; \
    } \
} \
#endif \
#else \
#ifdef DEBUG \
/* !NO_MPI DEBUG */ \
/* can use MPI_Abort() */ \
#define PCT_MALLOC(P, S, T) \
{ \
    if ((S) <= 0) \
    { \
        fprintf(stderr, \
            "ERROR: %s %d: proc %d: " \
            "malloc called for %ld bytes\n", \
            FILE_, LINE_, Myid, \
            (long) (S)); \
        MPI_Abort(MPI_COMM_WORLD, ABORT_INVALID_ARG); \
    } \
    if (!((P) = (T *) malloc((S)))) \

```

10

20

30

40

【表 1 5】

```

{
    fprintf(stderr,
        "ERROR: %s %d: proc %d: "
        "failed malloc %ld bytes\n",
        __FILE__, __LINE__, Myid,
        (long) (S));
    MPI_Abort(MPI_COMM_WORLD, ABORT_INVALID_IDX);
}
N_mallocs ++;
N_allocated_bytes += (S);
if (Next_alloc_idx == N_ALLOC_ENTRIES)
{
    fprintf(stderr,
        "ERROR: %s %d: proc %d: "
        "Next_alloc_idx %d (max %d)\n",
        __FILE__, __LINE__, Myid,
        Next_alloc_idx,
        N_ALLOC_ENTRIES-1);
    MPI_Abort(MPI_COMM_WORLD, ABORT_INVALID_IDX);
}
Allocs[Next_alloc_idx].p = (P);
Allocs[Next_alloc_idx].len = (S);
Next_alloc_idx ++;
N_outstanding_allocated_bytes += (S);
if (N_outstanding_allocated_bytes
    > N_allocated_bytes_high_water_mark)
{
    N_allocated_bytes_high_water_mark =
        N_outstanding_allocated_bytes;
}
if ((unsigned long) (S)
    > Largest_single_mem_request)
{
    Largest_single_mem_request = (S);
    strcpy(Largest_single_mem_request_fname, __FILE__);
    Largest_single_mem_request_line_no = __LINE__;
}
}
#else
/* !NO_MPI !DEBUG */
#define PCT_MALLOC(P, S, T)
{
    if (!(P) = (T *) malloc((S)))
    {
        fprintf(stderr,
            "ERROR: %s %d: proc %d: "
            "failed malloc %ld bytes\n",
            __FILE__, __LINE__, Myid,
            (long) (S));
        MPI_Abort(MPI_COMM_WORLD, ABORT_FAILED_MALLOC);
    }
    N_mallocs ++;
    N_allocated_bytes += (S);
    if ((unsigned long) (S)
        > Largest_single_mem_request)
    {
        Largest_single_mem_request = (S);
        strcpy(Largest_single_mem_request_fname, __FILE__);
        Largest_single_mem_request_line_no = __LINE__;
    }
}
#endif
#endif

```

【表 1 6】

```

#ifdef NO_MPI
#ifdef DEBUG
/* NO_MPI DEBUG */
/* must use assert(0) instead of MPI_Abort() */
#define PCT_REALLOC(P, S, T)
{
    int aidx;
    if ((S) <= 0)
    {
        fprintf(stderr,
            "ERROR: %s %d: proc %d: "
            "realloc called for %ld bytes\n",
            __FILE__, __LINE__, Myid,
            (long) (S));
        assert(0);
    }
    for (aidx = 0;
        aidx < Next_alloc_idx
        && Allocs[aidx].p != (P);
        aidx ++);
    ;
    if (aidx == Next_alloc_idx)
    {
        fprintf(stderr,
            "WARNING: %s %d: proc %d: "
            "free() could not find addr %p\n",
            __FILE__, __LINE__, Myid,
            (void *) (P));
    }
    N_outstanding_allocated_bytes -= Allocs[aidx].len;
    N_freed_bytes += Allocs[aidx].len;
    N_frees ++;
    if (!(P) == (T *) realloc((P), (S)))
    {
        fprintf(stderr,
            "ERROR: %s %d: proc %d: "
            "failed realloc %ld bytes\n",
            __FILE__, __LINE__, Myid,
            (long) (S));
        assert(0);
    }
    Allocs[aidx].p = (P);
    Allocs[aidx].len = (S);
    N_reallocs ++;
    N_allocated_bytes += (S);
    N_outstanding_allocated_bytes += (S);
    if (N_outstanding_allocated_bytes
        > N_allocated_bytes_high_water_mark)
    {
        N_allocated_bytes_high_water_mark =
            N_outstanding_allocated_bytes;
    }
    if ((unsigned long) (S)
        > Largest_single_mem_request)
    {
        Largest_single_mem_request = (S);
        strcpy(Largest_single_mem_request_fname, __FILE__);
        Largest_single_mem_request_line_no = __LINE__;
    }
}
#else
/* NO_MPI !DEBUG */
#define PCT_REALLOC(P, S, T)
{
    if (!(P) == (T *) realloc((P), (S)))

```

【表 1 7】

```

{
    fprintf(stderr,
        "ERROR: %s %d: proc %d: "
        "failed realloc %ld bytes\n",
        __FILE__, __LINE__, Myid,
        (long) (S));
    assert(0);
}
N_reallocs ++;
N_allocated_bytes += (S);
if ((unsigned long) (S)
    > Largest_single_mem_request)
{
    Largest_single_mem_request = (S);
    strcpy(Largest_single_mem_request_fname, __FILE__);
    Largest_single_mem_request_line_no = __LINE__;
}
}
#endif
#else
#ifdef DEBUG
/* !NO_MPI_DEBUG */
/* can use MPI_Abort() */
#define PCT_REALLOC(P, S, T)
{
    int aidx;
    if ((S) <= 0)
    {
        fprintf(stderr,
            "ERROR: %s %d: proc %d: "
            "realloc called for %ld bytes\n",
            __FILE__, __LINE__, Myid,
            (long) (S));
        MPI_Abort(MPI_COMM_WORLD, ABORT_INVALID_ARG);
    }
    for (aidx = 0;
        aidx < Next_alloc_idx
        && Allocs[aidx].p != (P);
        aidx++)
    ;
    if (aidx == Next_alloc_idx)
    {
        fprintf(stderr,
            "WARNING: %s %d: proc %d: "
            "free() could not find addr %p\n",
            __FILE__, __LINE__, Myid,
            (void *) (P));
    }
    N_outstanding_allocated_bytes -= Allocs[aidx].len;
    N_freed_bytes += Allocs[aidx].len;
    N_frees ++;
    if (!(P) == (T *) realloc((P), (S)))
    {
        fprintf(stderr,
            "ERROR: %s %d: proc %d: "
            "failed realloc %ld bytes\n",
            __FILE__, __LINE__, Myid,
            (long) (S));
        MPI_Abort(MPI_COMM_WORLD, ABORT_FAILED_MALLOC);
    }
    Allocs[aidx].p = (P);
    Allocs[aidx].len = (S);
    N_reallocs ++;
    N_allocated_bytes += (S);
    N_outstanding_allocated_bytes += (S);
}

```

【表 18】

```

if (N_outstanding_allocated_bytes
    > N_allocated_bytes_high_water_mark)
{
    N_allocated_bytes_high_water_mark =
        N_outstanding_allocated_bytes;
}
if ((unsigned long) (S)
    > Largest_single_mem_request)
{
    Largest_single_mem_request = (S);
    strcpy(Largest_single_mem_request_fname, __FILE__);
    Largest_single_mem_request_line_no = __LINE__;
}
}
#else
/* !NO_MPI !DEBUG */
#define PCT_REALLOC(P, S, T)
{
    if (!(P) = (T *) realloc((P), (S)))
    {
        fprintf(stderr,
            "ERROR: %s %d: proc %d: "
            "failed realloc %ld bytes\n",
            __FILE__, __LINE__, Myid,
            (long) (S));
        MPI_Abort(MPI_COMM_WORLD, ABORT_FAILED_MALLOC);
    }
    N_reallocs ++;
    N_allocated_bytes += (S);
    if ((unsigned long) (S)
        > Largest_single_mem_request)
    {
        Largest_single_mem_request = (S);
        strcpy(Largest_single_mem_request_fname, __FILE__);
        Largest_single_mem_request_line_no = __LINE__;
    }
}
#endif
#endif

#ifdef DEBUG
#define PCT_FREE(P)
{
    int aidx;
    for (aidx = 0;
        aidx < Next_alloc_idx
        && Allocs[aidx].p != (P);
        aidx ++);
    ;
    if (aidx == Next_alloc_idx)
    {
        fprintf(stderr,
            "WARNING: %s %d: proc %d: "
            "free() could not find addr %p\n",
            __FILE__, __LINE__, Myid,
            (void *) (P));
    }
    N_outstanding_allocated_bytes -= Allocs[aidx].len;
    N_freed_bytes += Allocs[aidx].len;
    Allocs[aidx].p = NULL;
    N_frees ++;
    free((P));
}
#else
#define PCT_FREE(P) free((P));

```

【表 1 9】

```

#endif

/*****
/* Data Structures */
*****/

// For space carving
// A high energy trace
#define HIGH_TRACE (1<<0)
// A low energy trace
#define LOW_TRACE (1<<1)
// All energy levels turned on
#define ALL_TRACE (HIGH_TRACE | LOW_TRACE)

struct entry_exit_t
{
    float entry_x, entry_y, entry_z, exit_x, exit_y, exit_z;
};

struct stats
{
    float mean;
    float sse;
    unsigned int n;
#ifdef DEBUG
    /* for sanity checking only, not needed for stats */
    float min, max;
#endif
};

/* partitioning internal exit detector plane into regions */
const int REGION_ROWS = 30;
const int REGION_COLS = 30;

const int STAT_DELTA_X = 0;
const int STAT_DELTA_Y = 1;
const int STAT_DELTA_ENERGY = 2;
const int STAT_LAST_UNUSED = 3;
const int STAT_NSTATS = STAT_LAST_UNUSED;
/* linearized layout - from fastest changing idx -> slowest changing idx */
/* x,y,energy {0,1,2} -> REGION_COLS -> REGION_ROWS -> N_FIRING_ANGLES */
const int N_STAT_CELLS = (STAT_NSTATS
    * REGION_ROWS * REGION_COLS
    * N_FIRING_ANGLES);

const int STAT_CELL_IDX_ANGLE_STRIDE = (STAT_NSTATS
    * REGION_ROWS * REGION_COLS);
const int STAT_CELL_IDX_ROW_STRIDE = (STAT_NSTATS * REGION_COLS);
const int STAT_CELL_IDX_COL_STRIDE = STAT_NSTATS;

const float MAX_SIGMA = 3.0;

/* Voxel level debugging */
/* This is to help isolate calculations that */
/* work with a specific voxel in the data space. */

/* The particular voxel index to be examined. The constants below
   refer to the plane, row, and column number respectively. */

#ifdef DEBUG_VOXEL
#define DEBUG_VOXEL_IDX (225 * (MAX_ROWS*MAX_COLS) + 201 * MAX_COLS + 42)
#endif

#ifdef NO_MPI

```

【表 20】

```

/* MPI Profiling */
/* MPI functions we profile */
const int MPI_INIT_IDX      = 0;
const int MPI_COMM_SIZE_IDX = 1;
const int MPI_COMM_RANK_IDX = 2;
const int MPI_ADDRESS_IDX   = 3;
const int MPI_TYPE_STRUCT_IDX = 4;
const int MPI_TYPE_COMMIT_IDX = 5;
const int MPI_OP_CREATE_IDX  = 6;
const int MPI_OP_FREE_IDX    = 7;
/* put msg passing ones at end for pretty report */
const int MPI_ALLREDUCE_IDX  = 8;
const int MPI_BCAST_IDX     = 9;
const int MPI_SEND_IDX      = 10;
const int MPI_RECV_IDX      = 11;
const int MPI_LASTUNUSED_IDX = 12;
const int Nfuncs            = MPI_LASTUNUSED_IDX;

struct func_profile
{
    char func_name[50];
    unsigned call_count;
    double total_time_secs;
    int does_communicate;
    unsigned long total_bytes;
}; /* end struct func_profile */

EXTERN struct func_profile func_profiles[Nfuncs];
#endif

/* Input/Output data location */
/* PADS */
/* const char InputDirBase[] = "/home/karonis/38M/Input"; */
const char InputDirBase[] = "/gpfs/pads/projects/CI-CCR000048/karonis/Input";
/* const char InputDirBase[] = "/home/karonis/pCT/"; */
/* const char OutputDirBase[] = "/home/karonis/38M/Output"; */
const char OutputDirBase[] = "/gpfs/pads/projects/CI-CCR000048/karonis";
/* const char OutputDirBase[] = "/autofs/home/karonis/pCT/Code/"; */
/* Mac */
/* const char InputDirBase[] = "/Volumes/Software/pCT/"; */
/* const char OutputDirBase[] = "/Volumes/Software/pCT/Code/"; */
/* laptop */
/* const char InputDirBase[] = "/Users/karonis/pCT/"; */
/* const char OutputDirBase[] = "/Users/karonis/pCT/Code/"; */
/* hopper.cs.niu.edu */
/* const char InputDirBase[] = "/home/hopper/research/pct/data/SimProjections/Input"; */
/* const char OutputDirBase[] = "/home/turing/karonis/pCT/Code/"; */

#endif /* PCT_PCT_H */
#define __MAIN__

#include "pct.h"

/*****
/* LOCAL GLOBAL VARIABLES */
*****/

/* for linear solver loop */
const int Niters = 10;
static int Iterations = 0;

static char InputDir[256], InputFnamePrefix[50], InputFnameSuffix[50];
static unsigned TotalHistories; // relevant for foreman only

```

【表 2 1】

```

const float MB = 1024.0*1024.0;
const float GB = MB*1024.0;

/* Timer Indexes and count */
const int SetupTimeIdx      = 0;
const int IEDTimeIdx        = 1;
const int VolumeBitmaskTimeIdx = 2;
const int TouchedPhantomTimeIdx = 3;
const int BinnedCleanupIdx   = 4;
const int AdjustIEDIdx       = 5;
const int MLPTimeIdx         = 6;
const int LinearSolverTimeIdx = 7;
const int LastUnusedIdx      = 8;
const int Ntimers            = LastUnusedIdx;
suseconds_t Timers[Ntimers]  = {0};
unsigned long StartNhistories[Ntimers];
const char *TimerNames[Ntimers] =
{
    "Set up",
    "IED",
    "Volume Bitmask",
    "Connect to Phantom",
    "Binned cleanup",
    "Adjust IED",
    "MLP",
    "Linear Solver"
};

/*****
/* LOCAL FUNCTIONS */
*****/

static void accumulate_stats_val(float *m,
                                float *s,
#ifdef DEBUG
                                float *min,
                                float *max,
#endif
                                unsigned int *n,
                                float val);
static void compute_and_bcast_vbm(int my_nhistories,
                                unsigned char *history_bitmask,
                                float *input_floats,
                                unsigned short *input_ushorts,
                                int volume_bitmask_len,
                                char *volume_bitmask);
static void compute_nhistories(int *my_nhistories,
                                int *histories_per_worker, // relevant foreman only
                                int n_input_histories,
                                int max_histories_per_foreman_or_worker,
                                int *must_fill_foreman);
static float compute_stat_val(float *input_floats,
                                unsigned short *input_ushorts,
                                int h,
                                int stat_displ);
static int count_histories_in_bitmask(int my_nhistories,
                                unsigned char *history_bitmask);
static void fill_buffers_from_files(float *input_floats,
                                unsigned short *input_ushorts,
                                int *deg,
                                FILE **fp,
                                int *histories_read_from_this_file,
                                int max_histories_per_input_file,
                                int nhistories_buffsize,
                                int *my_nhistories,

```

【表 2 2】

```

        int must_fill);
static void find_entry_exit_from_vbm_or_remove (int nhistories,
        unsigned char *history_bitmask,
        float *input_floats,
        unsigned short *input_ushorts,
        struct entry_exit_t *entry_exit,
        int volume_bitmask_len,
        char *volume_bitmask);

static int get_stat_cell_idx(float *input_floats,
        unsigned short *input_ushorts,
        int h,
        int stat_displ);
static FILE *open_input_file(int degree);
static void parse_command_line_args(int argc,
        char *argv[],
        int *n_input_histories,
        int *max_histories_per_foreman_or_worker,
        int *max_histories_per_input_file,
        float *linear_solver_relaxation);
static void read_and_distribute_input(int max_histories_per_foreman_or_worker,
        int max_histories_per_input_file,
        int *my_nhistories,
        int must_fill_foreman,
        float *input_floats,
        unsigned short *input_ushorts);
static void remove_based_on_binned_stat(int nhistories,
        unsigned char *history_bitmask,
        float *input_floats,
        unsigned short *input_ushorts,
        struct stats s[],
        float sigma_cutoff[],
        int stat_displ);
static void remove_histories_from_binned_stats(int my_nhistories,
        unsigned char *history_bitmask,
        float *input_floats,
        unsigned short *input_ushorts);
static int test_solution_for_done(float *solution);

static inline void record_start_nhistories(unsigned long nhistories,
        const int idx,
        unsigned long my_nhistories)
{
    StartNhistories[idx] = nhistories;
    printf("%d: %18s: starting nhistories %ld - %% my N histories %5.1f %%\n",
        Myid, TimerNames[idx], StartNhistories[idx],
        (my_nhistories
        ? ((float) nhistories)*100.0/((float) my_nhistories)
        : 0.0 ));
} /* end record_starting_nhistories() */

// assumes after.tv_sec >= before.tv_sec
//
// there are only two cases ...
// Case I: after.tv_usec >= before.tv_usec (no matter how many secs passed)
// elapsed usecs = (after.tv_sec - before.tv_sec)*1,000,000
//                + (after.tv_usec - before.tv_usec)
//
// Case II: after.tv_usec < before.tv_usec (in this case after.tv_sec must be
//                greater than before.tv_sec)
// elapsed usecs = (after.tv_sec - before.tv_sec)*1,000,000
//                - (before.tv_usec - after.tv_usec)
//                = (after.tv_sec - before.tv_sec)*1,000,000
//                + after.tv_usec - before.tv_usec
//
// BOTH CASES ARE THE SAME

```

【0 0 9 5】

【表 2 3】

```

static inline void record_time(struct timeval &before,
                               struct timeval &after,
                               const int idx)
{
    Timers[idx] = (after.tv_sec - before.tv_sec)*1000000
                  + after.tv_usec - before.tv_usec;
} /* end record_time() */

#ifdef NO_MPI
/* need to write this because MPI_LAND in */
/* MPI_Reduce is _not_ defined for MPI_CHAR */
/* must be externally visable for MPI */
void andem(void *operand_vec,
            void *operand_result_vec,
            int *len,
            MPI_Datatype *dtype)
{
    char *op      = (char *) operand_vec;
    char *op_res  = (char *) operand_result_vec;
    int i;

    for (i = 0; i < *len; i++)
        op_res[i] &= op[i];

    return;
} /* end andem() */

/* operator for MPI_Allreduce() */
/*
"Updating Formulae and a Pairwise Algorithm for Computer Sample Variances,"
T.F. Chan, G.H. Golub, and R.J. LeVeque,
Technical Report STAN-CS-79-773, Department of Computer Science,
Stanford University, 1979,
ftp://reports.stanford.edu/pub/cstr/reports/cs/tr/79/773/CS-TR-79-773.pdf

Given two sample sets where the mean and SSE have been computed
for each; <M_k,1 S_k,1 N_1> and <M_k,2 S_k,2 N_2>
we can compute the combined mean and SSE as follows

N = N_1 + N_2
d = M_k,2 - M_k,1

combined mean = M_k,1 + d*N_2/N
combined SSE  = S_k,1 + S_k,2 + d^2*N_1*N_2/N

*/

void combine_stats(void *operand_vec,
                  void *operand_result_vec,
                  int *len,
                  MPI_Datatype *dtype)
{
    struct stats *op      = (struct stats *) operand_vec;
    struct stats *op_res  = (struct stats *) operand_result_vec;
    int i;
    unsigned int combined_n;
    float delta;

    for (i = 0; i < *len; i++)
    {
        if (op[i].n > 0 && op_res[i].n > 0)
        {
            combined_n = op[i].n + op_res[i].n;

```

【表 2 4】

```

        delta      = op[i].mean - op_res[i].mean;

        op_res[i].mean += (delta * op[i].n / combined_n);
        op_res[i].sse  += (op[i].sse +
                           (delta * delta * op[i].n * op_res[i].n / combined_n));
        op_res[i].n    = combined_n;
#ifdef DEBUG
        if (op_res[i].min > op[i].min) op_res[i].min = op[i].min;
        if (op_res[i].max < op[i].max) op_res[i].max = op[i].max;
#endif
    }
    else if (op[i].n > 0 && op_res[i].n == 0)
    {
        op_res[i].n    = op[i].n;
        op_res[i].mean = op[i].mean;
        op_res[i].sse  = op[i].sse;
#ifdef DEBUG
        op_res[i].min  = op[i].min;
        op_res[i].max  = op[i].max;
#endif
    } /* endif */
} /* endfor */

return;

} /* end combine_stats() */
#endif

/*****
/* EXTERNAL FUNCTIONS */
*****/

/* IED func */
//extern void compute_IED(int my_nhistories, float *input_floats, float *IED);
#include "ied.h"

// intersect2.cc
extern bool compute_entry_exit_from_vbm(int h,
                                         float *input_floats,
                                         unsigned short *input_ushorts,
                                         int volume_bitmask_len,
                                         char *volume_bitmask,
                                         struct entry_exit_t *entry_exit);
extern void compute_initial_vbm(int my_nhistories,
                                unsigned char *history_bitmask,
                                float *input_floats,
                                unsigned short *input_ushorts,
                                int volume_bitmask_len,
                                char *volume_bitmask);
extern void morph_clean(int volume_bitmask_len, char *volume_bitmask);
/* MLP func */
extern void compute_mlp(int my_nhistories,
                        unsigned char *history_bitmask,
                        float *input_floats,
                        unsigned short *input_ushorts,
                        struct entry_exit_t *entry_exit,
                        int volume_bitmask_len,
                        char *volume_bitmask,
                        MLP_m *mlp);

/*****
/* Linear solvers */
*****/

```

【0 0 9 7】

【表 2 5】

```

/* string averaging methods */
extern void carp(MLP_m *, float x[], float b[],
                int count[], unsigned char history_bitmask[],
                int n_rows, int n_cols, float relax);
extern void sap(MLP_m *, float x[], float b[],
                int count[], unsigned char history_bitmask[],
                int n_rows, int n_cols, float relax);

typedef void (* linear_solver_func ) (MLP_m *,
                                     float x[],
                                     float b[],
                                     int count[],
                                     unsigned char history_bitmask[],
                                     int n_rows,
                                     int n_cols,
                                     float relax);

const int CARP_IDX = 0;
const int SAP_IDX = 1;
linear_solver_func linear_solvers[] = {carp, sap};

int main(int argc, char *argv[])
{
    int i;
    float *input_floats;
    unsigned short *input_ushorts;
    int my_nhistories;
    int n_input_histories, max_histories_per_foreman_or_worker;
    int histories_per_worker; // relevant for foreman only
    int max_histories_per_input_file; // relevant for foreman only
    unsigned long linear_solver_iterations = 0; // relevant for foreman only
#ifdef DEBUG
    unsigned long nvoxels_inside_phantom = 0; // for foreman only
    unsigned long nvoxels_outside_phantom = 0; // for foreman only
    unsigned long nvoxels_inside_phantom_mlp_touched = 0; // for foreman only
    unsigned long nvoxels_outside_phantom_mlp_touched = 0; // for foreman only
    unsigned long nvoxels_inside_phantom_not_touched = 0; // for foreman only
    unsigned long nvoxels_outside_phantom_not_touched = 0; // for foreman only
#endif
    unsigned long start_n_hist;
    int must_fill_foreman;
    /* int volume_bitmask_len = MAX_ROWS*MAX_COLS*MAX_PLANES; */
    int volume_bitmask_len = NVOXELS;
    char *volume_bitmask;
    unsigned char *history_bitmask;
    int history_bitmask_len;
    float *IED;
    struct entry_exit_t *entry_exit;
    MLP_m *mlp;
    float linear_solver_relaxation;
    float *solution;
    int *count;
    int linear_solver_idx = CARP_IDX;

    struct timeval before, after;
    time_t total_time_start, total_time_secs;

    total_time_start = time(NULL); // Mark the start time

#ifdef NO_MPI
    Nprocs = 1;
    Myid = 0;
#else
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &Nprocs);

```

【表 2 6】

```

    MPI_Comm_rank(MPI_COMM_WORLD, &Myid);
#endif

    /*
    /*****
    /* Set up */
    *****/

    record_start_nhistories(0, SetupTimeIdx, 0);
    gettimeofday(&before, NULL);
    {
        N_mallocs = N_reallocs = N_allocated_bytes
        = Largest_single_mem_request = 0;
#ifdef DEBUG
        N_frees = Next_alloc_idx = N_freed_bytes =
        N_outstanding_allocated_bytes = N_allocated_bytes_high_water_mark = 0;
#endif

        if (Myid == ForemanRank) printf("%d: nprocs %d\n", Myid, Nprocs);

        if (Myid == ForemanRank)
        {
            /* recording current time to embed in output fnames */
            time_t now = time(NULL);
            struct tm *tp = localtime(&now);

            sprintf(Nowstring, "%4d-%02d-%02d_%02d:%02d:%02d",
                    1900+tp->tm_year,
                    1+tp->tm_mon,
                    tp->tm_mday,
                    1+tp->tm_hour,
                    1+tp->tm_min,
                    1+tp->tm_sec);
        } /* endif */

#ifdef NO_MPI
        MPI_Bcast(&Nowstring, // buff
                100, // count
                MPI_CHAR, // data type
                ForemanRank, // root
                MPI_COMM_WORLD);
#endif

        parse_command_line_args(argc,
                                argv,
                                &n_input_histories,
                                &max_histories_per_foreman_or_worker,
                                &max_histories_per_input_file,
                                &linear_solver_relaxation);
        sprintf(Relaxparamstring, "relaxparam-%4.2f", linear_solver_relaxation);

        compute_nhistories(&my_nhistories,
                            &histories_per_worker,
                            n_input_histories,
                            max_histories_per_foreman_or_worker,
                            &must_fill_foreman);

        /* at this point: */
        /* - for all work procs: my_nhistories == histories_per_worker */
        /* - for foreman: my_nhistories >= histories_per_worker */
        mlp = new MLP_m(my_nhistories);
        /* fprintf(stderr, "%d: after new MLP_m::MLP_m(): curr_mem_block_idx %d n_mem_blocks %d\n", Myid, mlp->get_curr_mem_block_idx(), mlp->get_n_mem_blocks()); */

        if (Myid == ForemanRank)
            printf("%d: Input: %s/%s*%s\n",

```

【表 27】

```

        Myid, InputDir, InputFnamePrefix, InputFnameSuffix);

    PCT_MALLOC(input_floats,
                my_nhistories
                *INPUT_FLOAT_STRIDE
                *sizeof(float),
                float)
    PCT_MALLOC(input_ushorts,
                my_nhistories
                *INPUT_USHORT_STRIDE
                *sizeof(unsigned short),
                unsigned short)

    read_and_distribute_input(histories_per_worker,
                              max_histories_per_input_file,
                              &my_nhistories,
                              must_fill_foreman,
                              input_floats,
                              input_ushorts);

    if (Myid == ForemanRank)
        printf("%d: my_nhistories %d\n", Myid, my_nhistories);

    history_bitmask_len = (my_nhistories/8) + (my_nhistories % 8 ? 1 : 0);
    PCT_MALLOC(history_bitmask, history_bitmask_len, unsigned char)
    /* initially claim all histories are used in MLP */
    memset(history_bitmask, 0xff, history_bitmask_len);

}
gettimeofday(&after, NULL);
record_time(before, after, SetupTimeIdx);
/* fprintf(stderr, "%d: after setup: curr_mem_block_idx %d n_mem_blocks %d\n", Myid, ml
p->get_curr_mem_block_idx(), mlp->get_n_mem_blocks()); */

/*****
/* gentlemen, start your engines .. */
/* ... computing starts HERE */
*****/

/*****/
/* IED - 'b' in Ax=b */
/*****/
/* compute IED for all histories */
record_start_nhistories(my_nhistories, IEDTimeIdx, my_nhistories);
gettimeofday(&before, NULL);
/* if (0) */
{
    PCT_MALLOC(IED, my_nhistories*sizeof(float), float)
    compute_IED(my_nhistories, input_floats, IED);
}
gettimeofday(&after, NULL);
record_time(before, after, IEDTimeIdx);
/* fprintf(stderr, "%d: after IED: curr_mem_block_idx %d n_mem_blocks %d\n", Myid, mlp-
>get_curr_mem_block_idx(), mlp->get_n_mem_blocks()); */

/*****/
/* volume bit mask */
/*****/
start_n_hist = count_histories_in_bitmask(my_nhistories, history_bitmask);
record_start_nhistories(start_n_hist, VolumeBitmaskTimeIdx, my_nhistories);
gettimeofday(&before, NULL);
/* if (0) */
{
    PCT_MALLOC(volume_bitmask, volume_bitmask_len, char)
    compute_and_bcast_vbm(my_nhistories,

```

【0100】

【表 28】

```

        history_bitmask,
        input_floats,
        input_ushorts,
        volume_bitmask_len,
        volume_bitmask);
    }
    gettimeofday(&after, NULL);
    record_time(before, after, VolumeBitmaskTimeIdx);
    /* fprintf(stderr, "%d: after VBM: curr_mem_block_idx %d n_mem_blocks %d\n", Myid, mlp-
    >get_curr_mem_block_idx(), mlp->get_n_mem_blocks()); */

if (0)
{
    if (Myid == ForemanRank)
    {
        FILE *fp;
        char fname[300];

        sprintf(fname, "%s/volume_bitmask.%s.len.%d",
            OutputDirBase, Nowstring, volume_bitmask_len);
        if ((fp = fopen(fname, "w")) != NULL)
        {
            for (i = 0; i < volume_bitmask_len; i++)
                fprintf(fp, "%d\n", volume_bitmask[i]);

            fclose(fp);
        }
        else
        {
            fprintf(stderr, "ERROR: foreman could not fopen >%s<\n", fname);
        } /* endif */
        /* printf("%d: foreman should print solution here\n", Myid); */
    } /* endif */
}

/*****
/* remove remaining bad histories before MLP calculation */
*****/

PCT_MALLOC(entry_exit,
    my_nhistories*sizeof(struct entry_exit_t),
    struct entry_exit_t);

/* removing histories that don't touch phantom */
start_n_hist = count_histories_in_bitmask(my_nhistories, history_bitmask);
record_start_nhistories(start_n_hist, TouchedPhantomTimeIdx, my_nhistories);
gettimeofday(&before, NULL);
/* if (0) */
{
    /* populates entry_exit */

    find_entry_exit_from_vbm_or_remove(my_nhistories,
        history_bitmask,
        input_floats,
        input_ushorts,
        entry_exit,
        volume_bitmask_len,
        volume_bitmask);
}
gettimeofday(&after, NULL);
record_time(before, after, TouchedPhantomTimeIdx);

/* removing histories that are statistical outliers */
start_n_hist = count_histories_in_bitmask(my_nhistories, history_bitmask);

```

10

20

30

40

【0 1 0 1】

【表 2 9】

```

record_start_nhistories(start_n_hist, BinnedCleanupIdx, my_nhistories);
gettimeofday(&before, NULL);
/* if (0) */
{
    /* printf("%d: before remove_histories_from_binned_stats()\n", Myid); */
    remove_histories_from_binned_stats(my_nhistories,
                                        history_bitmask,
                                        input_floats,
                                        input_ushorts);
    /* printf("%d: after remove_histories_from_binned_stats()\n", Myid); */
}
gettimeofday(&after, NULL);
record_time(before, after, BinnedCleanupIdx);
/* fprintf(stderr, "%d: after remove remaining: curr_mem_block_idx %d n_mem_blocks %d\n",
   Myid, mlp->get_curr_mem_block_idx(), mlp->get_n_mem_blocks()); */

/*****/
/* Adjust IED */
/*****/
start_n_hist = count_histories_in_bitmask(my_nhistories, history_bitmask);
record_start_nhistories(start_n_hist, AdjustIEDIdx, my_nhistories);
gettimeofday(&before, NULL);
/* if (0) */
{
    adjust_IED(my_nhistories, history_bitmask, input_ushorts, input_floats,
               IED, entry_exit);
}
gettimeofday(&after, NULL);
record_time(before, after, AdjustIEDIdx);
/* fprintf(stderr, "%d: after Adjust IED: curr_mem_block_idx %d n_mem_blocks %d\n",
   Myid, mlp->get_curr_mem_block_idx(), mlp->get_n_mem_blocks()); */

/*****/
/* MLP - 'A' in Ax = b */
/*****/
start_n_hist = count_histories_in_bitmask(my_nhistories, history_bitmask);
record_start_nhistories(start_n_hist, MLPTIMEIdx, my_nhistories);
gettimeofday(&before, NULL);
/* if (0) */
{
    mlp->set_my_nhistories_for_MLP(count_histories_in_bitmask(my_nhistories,
                                                             history_bitmask));
    compute_mlp(my_nhistories,
                history_bitmask,
                input_floats,
                input_ushorts,
                entry_exit,
                volume_bitmask_len,
                volume_bitmask,
                mlp);
}
gettimeofday(&after, NULL);
record_time(before, after, MLPTIMEIdx);
/* fprintf(stderr, "%d: after MLP: curr_mem_block_idx %d n_mem_blocks %d\n", Myid, mlp-
   >get_curr_mem_block_idx(), mlp->get_n_mem_blocks()); */

PCT_FREE(entry_exit);

PCT_FREE(input_ushorts);
PCT_FREE(input_floats);

/*****/
/* solve for 'x' in Ax = b */
/*****/
start_n_hist = count_histories_in_bitmask(my_nhistories, history_bitmask);

```

【表 3 0】

```

record_start_nhistories(start_n_hist, LinearSolverTimeIdx, my_nhistories);
gettimeofday(&before, NULL);
/* if (0) */
{
    char done = 0;
#ifdef NO_MPI
    int *dummy_i_recv_buff = (int *) NULL; // needed for bug in MVAPICH2
    float *dummy_f_recv_buff = (float *) NULL; // needed for bug in MVAPICH2
#endif

    PCT_MALLOC(solution, NVOXELS*sizeof(float), float)
    PCT_MALLOC(count, NVOXELS*sizeof(int), int)

    /* seed solution with zeros */
    /* bzero(solution, NVOXELS*sizeof(float)); */
    /* seed solution in phantom with ones and outside phantom 0.001 */
    for (i = 0; i < NVOXELS; i++)
    {
        if (volume_bitmask[i])
        {
            solution[i] = INITIAL_SOLUTION_INSIDE_PHANTOM;
#ifdef DEBUG
            if (Myid == ForemanRank)
                nvoxels_inside_phantom++;
#endif
        }
        else
        {
            solution[i] = INITIAL_SOLUTION_OUTSIDE_PHANTOM;
#ifdef DEBUG
            if (Myid == ForemanRank)
                nvoxels_outside_phantom++;
#endif
        } /* endif */
    } /* endfor */

    while (!done)
    {
        linear_solver_iterations++;
        /* printf("%d: starting linear solver iteration %ld\n", */
        /* Myid, linear_solver_iterations); */

        bzero(count, NVOXELS*sizeof(int));

        /* solve for 'x' in Ax = b */
        linear_solvers[linear_solver_idx](mlp, // A
                                          solution, // x
                                          IED, // b
                                          count,
                                          history_bitmask,
                                          my_nhistories,
                                          NVOXELS,
                                          linear_solver_relaxation);

#ifdef NO_MPI
        MPI_Reduce((Myid == ForemanRank
                    ? MPI_IN_PLACE // res clobbers oprnd in root
                    : solution), // operand for non-root
                  (Myid == ForemanRank
                    ? solution // result, relevant at root only
                    : dummy_f_recv_buff),
                  NVOXELS, // count
                  MPI_FLOAT, // data type
                  MPI_SUM, // operator
                  ForemanRank, // root
                  MPI_COMM_WORLD);

```

10

20

30

40

【0 1 0 3】

【表 3 1】

```

MPI_Reduce((Myid == ForemanRank
           ? MPI_IN_PLACE           // res clobbers oprnd in root
           : count),                // operand for non-root
           (Myid == ForemanRank
           ? count                    // result, relevant at root only
           : dummy_i_recv_buff),
           NVOXELS,                  // count
           MPI_INT,                  // data type
           MPI_SUM,                  // operator
           ForemanRank,              // root
           MPI_COMM_WORLD);

#endif

if (Myid == ForemanRank)
{
    /* NICK: not sure about count[i] == 0 here ... not sure */
    /*      if that's an error or not :-( */

#ifdef DEBUG
    if (linear_solver_iterations == 1)
    {
        /* volume_bitmask[] and count[] should be the same */
        /* for all iterations, so suffices to do this only */
        /* once and for the first iteration */
        for (i = 0; i < NVOXELS; i++)
        {
            if (count[i] > 0)
            {
                /* this voxel touched in MLP by >0 histories */
                if (volume_bitmask[i])
                {
                    nvoxels_inside_phantom_mlp_touched++;
                }
                else
                {
                    nvoxels_outside_phantom_mlp_touched++;
                } /* endif */
            }
            else
            {
                /* this voxel not touched in MLP by any histories */
                if (volume_bitmask[i])
                {
                    nvoxels_inside_phantom_not_touched++;
                }
                else
                {
                    nvoxels_outside_phantom_not_touched++;
                } /* endif */
            } /* endif */
        } /* endfor */
    } /* endif */

    for (i = 0; i < NVOXELS; i++)
    {
        if (count[i] > 0)
            solution[i] /= count[i];
        else
        {
            /* count[i] == 0 means that none of the histories, */
            /* from any proc, touched this voxel_idx=i ... */
            /* ... however, to make the MPI_Reduce sum for */
            /* count and solution_vector work easily, each proc */
            /* (inside linear solver) set their local copy of */

```

【0 1 0 4】

【表 3 2】

```

/* solution_vector[i] = 0 for those voxels=i that */
/* it determined was not touched by any of _its_ */
/* histories ... the result of all this is that */
/* if _none_ of the histories touched a voxel then */
/* its summed value in the foreman is now 0.0 and */
/* it needs to be set back to the initialization */
/* guess based on whether we think the voxel is in */
/* the phantom or not */

if (volume_bitmask[i])
{
    solution[i] = INITIAL_SOLUTION_INSIDE_PHANTOM;
}
else
{
    solution[i] = INITIAL_SOLUTION_OUTSIDE_PHANTOM;
} /* endif */
} /* endif */
} /* endfor */

done = test_solution_for_done(solution);
/* if (0) */
{
    if (!done)
    {
        FILE *fp;
        char fname[300];

        sprintf(fname, "%s/solution_vector.%s.%s.%02ld",
            OutputDirBase,
            Nowstring,
            Relaxparamstring,
            linear_solver_iterations);
        if ((fp = fopen(fname, "w")) != NULL)
        {
            fprintf(fp, "%d %d %d\n",
                MAX_COLS, MAX_ROWS, MAX_PLANES);
            for (i = 0; i < NVOXELS; i++)
                fprintf(fp, "%f\n", solution[i]);

            fclose(fp);
        }
        else
        {
            fprintf(stderr,
                "ERROR: foreman could not fopen >%s<\n", fname);
        } /* endif */
    } /* endif */
} /* endif */

#ifdef NO_MPI
// foreman bcast done
MPI_Bcast(&done, // buff
    1, // count
    MPI_CHAR, // data type
    ForemanRank, // root
    MPI_COMM_WORLD);

if (!done)
{
    // foreman bcast done
    MPI_Bcast(solution, // buff
        NVOXELS, // count
        MPI_FLOAT, // data type

```

【表 3 3】

```

        ForemanRank, // root
        MPI_COMM_WORLD);
    } /* endif */
#endif
} /* endwhile */
}
gettimeofday(&after, NULL);
record_time(before, after, LinearSolverTimeIdx);
/* fprintf(stderr, "%d: after linsolv: curr_mem_block_idx %d n_mem_blocks %d\n", Myid,
mlp->get_curr_mem_block_idx(), mlp->get_n_mem_blocks()); */

PCT_FREE(volume_bitmask);
PCT_FREE(count);
PCT_FREE(history_bitmask);
PCT_FREE(IED);

/* printing solution */
/* if (0) */
{
    if (Myid == ForemanRank)
    {
        FILE *fp;
        char fname[300];

        sprintf(fname, "%s/solution_vector.%s.%s.final",
            OutputDirBase, Nowstring, Relaxparamstring);
        if ((fp = fopen(fname, "w")) != NULL)
        {
            fprintf(fp, "%d %d %d\n",
                MAX_COLS, MAX_ROWS, MAX_PLANES);
            for (i = 0; i < NVOXELS; i++)
                fprintf(fp, "%f\n", solution[i]);

            fclose(fp);
        }
        else
        {
            fprintf(stderr, "ERROR: foreman could not fopen >%s<\n", fname);
        } /* endif */
        /* printf("%d: foreman should print solution here\n", Myid); */
    } /* endif */
}

PCT_FREE(solution);

/* print summary stats */
total_time_secs = time(NULL) - total_time_start;
/* Time */
printf("\n");
printf("%d: ----- Performance Summary\n", Myid);
#ifdef OPT
    printf("%d: Build Version: Optimized\n", Myid);
#endif
#ifdef NOOPT
    printf("%d: Build Version: Not optimized\n", Myid);
#endif
#ifdef DEBUG
    printf("%d: Build Version: Debug\n", Myid);
#endif
printf("%d: Input: %s/%s%s\n",
    Myid, InputDir, InputFnamePrefix, InputFnameSuffix);

if (Myid == ForemanRank)
{
    printf("%d: Output filename timestamp: %s\n", Myid, Nowstring);
}

```

10

20

30

40

【0 1 0 6】

【表 3 4】

```

if (max_histories_per_input_file
    == UNSPECIFIED_MAX_NINPUT_HISTORIES_PER_FILE)
    printf("%d: Max Hist per File: unspecified\n", Myid);
else
    printf("%d: Max Hist per File: %d\n",
        Myid, max_histories_per_input_file);

printf("%d: Total N histories:          %d\n", Myid, TotalHistories);
switch (linear_solver_idx)
{
    case CARP_IDX:
        printf("%d: Linear Solver Method:    CARP\n", Myid);
        break;
    case SAP_IDX:
        printf("%d: Linear Solver Method:    SAP\n", Myid);
        break;
    default:
        printf("%d: Linear Solver Method:    unknown :-(\n", Myid);
        break;
} /* end switch() */
printf("%d: Linear Solver Relaxation: %f\n",
    Myid, linear_solver_relaxation);
printf("%d: Linear Solver Iterations: %ld\n",
    Myid, linear_solver_iterations);
} /* endif */

printf("%d: My N histories:              %d\n",
    Myid, my_nhistories);

#ifdef HISTOGRAM
    printf("%d: GENERATED HISTOGRAM DATA\n", Myid);
#endif

/* if (0) */
{
    printf("%d: -- Timers\n", Myid);
    for (i = 0; i < Ntimers; i++)
    {
        printf("%d: %18s: usecs: %ld (%ld secs) nhistories %ld\n",
            Myid, TimerNames[i], (long) Timers[i],
            (long) Timers[i]/1000000,
            StartNhistories[i]);
        if (StartNhistories[i] != 0)
            printf("%d: %18s  usecs/history %ld\n",
                Myid, " ",
                (long) Timers[i]/StartNhistories[i]);
    } /* endfor */
    printf("%d: --\n", Myid);
    printf("%d: Total execution time (secs): %ld\n", Myid, total_time_secs);
    printf("%d:\n", Myid);
}

/* Memory */
/* if (0) */
{
    unsigned long nbytes_allocd_for_voxel_idx =
        mlp->get_nbytes_allocd_for_voxel_idx();
    unsigned long n_histories_for_mlp = mlp->get_my_nhistories_for_MLP();
    unsigned long n_recorded_voxel_idx = mlp->get_n_recorded_voxel_idx();
    unsigned long nbytes_for_recorded_voxel_idx =
        sizeof(unsigned int)*n_recorded_voxel_idx;
    int n_allocd_blocks = mlp->get_n_allocd_blocks();
    unsigned int n_bytes_block;

```

【 0 1 0 7 】

【表 3 5】

```

printf("%d: ----- Memory Summary\n", Myid);

printf("%d: -- mem for MLP voxel idx's\n", Myid);
printf("%d:      N bytes allocd for voxel idxs: "
       "%ld (%7.1f MB %4.1f GB)\n",
       Myid, nbytes_allocd_for_voxel_idx,
       ((float) nbytes_allocd_for_voxel_idx)/MB,
       ((float) nbytes_allocd_for_voxel_idx)/GB);
printf("%d:      N bytes used for voxel idxs: "
       "%ld (%7.1f MB %4.1f GB)\n",
       Myid, nbytes_for_recorded_voxel_idx,
       ((float) nbytes_for_recorded_voxel_idx)/MB,
       ((float) nbytes_for_recorded_voxel_idx)/GB);
printf("%d:      N bytes unused for voxel idxs: "
       "%ld (%7.1f MB %4.1f GB)\n",
       Myid, nbytes_allocd_for_voxel_idx-nbytes_for_recorded_voxel_idx,
       ((float) (nbytes_allocd_for_voxel_idx-nbytes_for_recorded_voxel_idx))/MB,
       ((float) (nbytes_allocd_for_voxel_idx-nbytes_for_recorded_voxel_idx))/GB);
printf("%d: Percent alloc used for voxel idxs: %4.1f %% \n",
       Myid,
       (nbytes_allocd_for_voxel_idx != 0
        ? 100.0*((float) nbytes_for_recorded_voxel_idx)
          /((float) nbytes_allocd_for_voxel_idx)
        : 0.0));

printf("%d: -- counts for MLP voxel idx's\n", Myid);
printf("%d: N recorded voxel idxs: %ld\n", Myid, n_recorded_voxel_idx);
printf("%d: N histories for MLP: %ld\n", Myid, n_histories_for_mlp);
printf("%d: Avg rec v_idx/hist: %ld\n",
       Myid,
       (n_histories_for_mlp != 0
        ? n_recorded_voxel_idx/n_histories_for_mlp
        : 0L));

#ifdef DEBUG
if (Myid == ForemanRank)
{
printf("%d: -- Voxel space\n", Myid);
printf("%d: N voxels inside phantom: %ld (%4.1f %%)\n",
       Myid, nvoxels_inside_phantom,
       (100.0*((float) nvoxels_inside_phantom)/((float) NVOXELS)));
printf("%d: N voxels outside phantom: %ld (%4.1f %%)\n",
       Myid, nvoxels_outside_phantom,
       (100.0*((float) nvoxels_outside_phantom)/((float) NVOXELS)));
printf("%d: N voxels MLP-touched inside phantom: "
       "%ld (%4.1f %% of inside voxels - want 100%%)\n",
       Myid, nvoxels_inside_phantom_mlp_touched,
       (100.0*((float) nvoxels_inside_phantom_mlp_touched)
        /((float) nvoxels_inside_phantom)));
printf("%d: N voxels MLP-touched outside phantom: "
       "%ld (should be zero)\n",
       Myid, nvoxels_outside_phantom_mlp_touched);
printf("%d: N voxels *not* MLP-touched inside phantom: "
       "%ld (%4.1f %% of inside voxels - want zero)\n",
       Myid, nvoxels_inside_phantom_not_touched,
       (100.0*((float) nvoxels_inside_phantom_not_touched)
        /((float) nvoxels_inside_phantom)));
printf("%d: N voxels *not* MLP-touched outside phantom: "
       "%ld (%4.1f %% of outside voxels - should 100%%)\n",
       Myid, nvoxels_outside_phantom_not_touched,
       (100.0*((float) nvoxels_outside_phantom_not_touched)
        /((float) nvoxels_outside_phantom)));
} /* endif */
#endif

printf("%d: -- Block stats\n", Myid);

```

【0108】

【表 3 6】

```

printf("%d:          N copies: %ld\n", Myid, mlp->get_n_copies());
printf("%d: N bytes copied: %ld\n", Myid, mlp->get_n_copied_bytes());
printf("%d: N allocd blocks: %d\n", Myid, n_allocd_blocks);
for (i = 0; i < n_allocd_blocks; i++)
{
    n_bytes_block = mlp->get_block_nbytes(i);
    printf("%d: - blk %2d N bytes: %d (%7.1f MB  %4.1f GB)\n",
        Myid, i, n_bytes_block,
        ((float) n_bytes_block)/MB,
        ((float) n_bytes_block)/GB);
} /* endif */

delete mlp;

printf("%d: -- Basic memory stats\n", Myid);
printf("%d:          N calls to malloc(): %ld\n", Myid, N_mallocs);
printf("%d:          N calls to realloc(): %ld\n", Myid, N_reallocs);
#ifdef DEBUG
printf("%d:          N calls to free(): %ld\n", Myid, N_frees);
#endif
printf("%d:          N allocated bytes: %ld (%7.1f MB  %4.1f GB)\n",
    Myid, N_allocated_bytes,
    ((float) N_allocated_bytes)/MB, ((float) N_allocated_bytes)/GB);
printf("%d: Largest single mem req bytes: %ld (%7.1f MB  %4.1f GB)\n",
    Myid, Largest_single_mem_request,
    ((float) Largest_single_mem_request)/MB,
    ((float) Largest_single_mem_request)/GB);
printf("%d:          : file: %s   line: %d\n",
    Myid, Largest_single_mem_request_fname,
    Largest_single_mem_request_line_no);
#ifdef DEBUG
printf("%d:          N freed bytes: %ld (%7.1f MB  %4.1f GB)\n",
    Myid, N_freed_bytes,
    ((float) N_freed_bytes)/MB, ((float) N_freed_bytes)/GB);
printf("%d:          N not freed bytes: %ld (%7.1f MB  %4.1f GB)\n",
    Myid, N_allocated_bytes-N_freed_bytes,
    ((float) (N_allocated_bytes-N_freed_bytes))/MB,
    ((float) (N_allocated_bytes-N_freed_bytes))/GB);
printf("%d:          High water mark bytes: %ld (%7.1f MB  %4.1f GB)\n",
    Myid, N_allocated_bytes_high_water_mark,
    ((float) N_allocated_bytes_high_water_mark)/MB,
    ((float) N_allocated_bytes_high_water_mark)/GB);
#endif

printf("\n");
}

#ifdef DEBUG
#ifdef NO_MPI
/* MPI Profiles */
/* if (0) */
{
    printf("%d: -- MPI Functions\n", Myid);
    for (i = 0; i < Nfuncs; i++)
    {
        if (func_profiles[i].does_communicate)
        {
            printf("%d: %15s: Ncalls %3d secs %f  bytes %ld (%6.1f GB/sec)\n",
                Myid,
                func_profiles[i].func_name,
                func_profiles[i].call_count,
                func_profiles[i].total_time_secs,
                func_profiles[i].total_bytes,
                (func_profiles[i].total_time_secs > 0.0
                 ? (func_profiles[i].total_bytes/GB)

```

【0109】

【表 3 7】

```

        /func_profiles[i].total_time_secs
        : 0.0)
        );
    }
    else
    {
        printf("%d: %15s: Ncalls %3d secs %f\n",
            Myid,
            func_profiles[i].func_name,
            func_profiles[i].call_count,
            func_profiles[i].total_time_secs);
    } /* endif */
} /* endfor */

printf("\n");
}
#endif
#endif

/*****
/* epilogue */
*****/

#ifdef NO_MPI
    MPI_Finalize();
#endif

    exit(0);

} /* end main() */

/*
"Note on a Method for Calculating Corrected Sums of Squares and Products,"
B.P. Welford, Technometrics, Vol 4, No. 3, Aug 1962, 419-420.

The Art of Computer Programming, D.E. Knuth, vol 2, 2nd Edition, p 216
The Art of Computer Programming, D.E. Knuth, vol 2, 3rd Edition, p 232

Reccurrence forumla that keeps a running mean (M_k) and running SSE (S_k)
for some stream of values X_i

For x_1
    M_1 = x_1
    S_1 = 0

For x_i i>1
    M_i = M_(i-1) + [(x_i - M_(i-1))/i]
    S_i = S_(i-1) + [(x_i - M_(i-1)) * (x_i - M_i)]

At any time k>1
    running mean = M_k
    running _sample_ (i.e., not _population_) variance = S_k/(k-1)
*/

static void accumulate_stats_val(float *m,
                                float *s,
                                float *min,
                                float *max,
                                unsigned int *n,
                                float val)
{
    *n = *n + 1;

```

【0 1 1 0】

【表 3 8】

```

        if (*n == 1)
        {
            *m = val;
            *s = 0.0;
#ifdef DEBUG
            *min = *max = val;
#endif
        }
        else
        {
            float old_mean = *m;
            float d = val - old_mean;

            *m += (d/(*n));
            *s += (d*(val - *m));
#ifdef DEBUG
            if (*min > val) *min = val;
            if (*max < val) *max = val;
#endif
        } /* endif */

        /* printf("after pushing %f RunningMean %f RunningSSE %f\n",val,*m,*s); */

        return;

    } /* end accumulate_stats_val() */

static void compute_and_bcast_vbm(int my_nhistories,
    unsigned char *history_bitmask,
    float *input_floats,
    unsigned short *input_ushorts,
    int volume_bitmask_len,
    char *volume_bitmask)
{
#ifdef NO_MPI
    MPI_Op op;
    char *dummy_recv_buff = (char *) NULL; // needed for bug in MVAPICH2
#endif

    // makevbm2.cc
    compute_initial_vbm(my_nhistories,
        history_bitmask,
        input_floats,
        input_ushorts,
        volume_bitmask_len,
        volume_bitmask);

#ifdef NO_MPI
    MPI_Op_create((MPI_User_function*) andem, // my user-defined func
        1, // commute flag
        &op); // op handle

    MPI_Reduce((Myid == ForemanRank
        ? MPI_IN_PLACE // result clobbers operand for root
        : volume_bitmask), // operand for non-root
        (Myid == ForemanRank
        ? volume_bitmask // result, relevant at root only
        : dummy_recv_buff),
        volume_bitmask_len, // count
        MPI_CHAR, // data type
        op, // operator
        ForemanRank, // root
        MPI_COMM_WORLD);

```

【表 3 9】

```

    MPI_Op_free(&op);
#endif

    // foreman only morph_clean,
    if (Myid == ForemanRank)
    {
        // morph_clean2.cc
        morph_clean(volume_bitmask_len, volume_bitmask);
    } /* endif */

#ifdef NO_MPI
    // foreman bcast morph_cleaned volume bitmask
    MPI_Bcast(volume_bitmask, // buff
              volume_bitmask_len, // count
              MPI_CHAR, // data type
              ForemanRank, // root
              MPI_COMM_WORLD);
#endif

#ifdef 0
    // intersect2.cc
    // NICK: push all this into mlp code so that you do NOT
    // have to malloc and store everything in entry_exit
    compute_entry_exit_from_vbm(my_nhistories,
                                input_floats,
                                input_ushorts,
                                entry_exit);
#endif

    return;

} /* end compute_and_bcast_vbm() */

static void compute_nhistories(int *my_nhistories,
                               int *histories_per_worker, // relevant foreman only
                               int n_input_histories,
                               int max_histories_per_foreman_or_worker,
                               int *must_fill_foreman)
{
    /* max_histories_per_foreman_or_worker is always defined */
    /* ... in that context there are two cases: */
    /* (1) we don't know how many input histories there are: */
    /* read max_histories_per_foreman_or_worker into */
    /* each process, must fill all workers, foreman is */
    /* last to get his and so may be less than full */
    /* (2) we know how many input histories there are: */
    /* spread histories evenly across all Nprocs, make */
    /* sure under max_histories_per_foreman_or_worker, */
    /* all workers get the same number of histories and */
    /* foreman may get more due to rounding */

    if (n_input_histories == UNSPECIFIED_NINPUT_HISTORIES)
    {
        *my_nhistories =
            *histories_per_worker = max_histories_per_foreman_or_worker;

        *must_fill_foreman = 0;
    }
    else
    {
        /* n_input_histories specified on command line */

        *histories_per_worker = n_input_histories/Nprocs;
        if (Myid == ForemanRank)

```

【表 4 0】

```

{
    /* Foreman must absorb rounding slack */
    *my_nhistories = n_input_histories -
                    ((Nprocs-1)*(*histories_per_worker));
}
else
{
    *my_nhistories = *histories_per_worker;
} /* endif */

*must_fill_foreman = 1;
} /* endif */

if (*my_nhistories > max_histories_per_foreman_or_worker)
{
    fprintf(stderr,
        "ERROR: %s %d: proc %d: "
        "my_nhistories %d > max_histories_per_foreman_or_worker %d\n",
        __FILE__, __LINE__, Myid,
        *my_nhistories, max_histories_per_foreman_or_worker);
    PCT_ABORT(ABORT_TOO_MANY_HISTORIES)
} /* endif */

return;

} /* end compute_nhistories() */

static float compute_stat_val(float *input_floats,
                              unsigned short *input_ushorts,
                              int h,
                              int stat_displ)
{
    float val;

    switch (stat_displ)
    {
        case STAT_DELTA_X:
        {
            int x1 = get_x1(input_ushorts, h);
            int x2 = get_x2(input_ushorts, h);
            int x3 = get_x3(input_ushorts, h);
            int x4 = get_x4(input_ushorts, h);

            val = ((float) (x4 - x3) - (x2 - x1));
        }
        break;
        case STAT_DELTA_Y:
        {
            int y1 = get_y1(input_ushorts, h);
            int y2 = get_y2(input_ushorts, h);
            int y3 = get_y3(input_ushorts, h);
            int y4 = get_y4(input_ushorts, h);

            val = ((float) (y4 - y3) - (y2 - y1));
        }
        break;
        case STAT_DELTA_ENERGY:
        {
            float ein = get_ein(input_floats, h);
            float eout = get_eout(input_floats, h);

            val = ein-eout;
        }
    }
}

```

【0 1 1 3】

【表 4 1】

```

        break;
default:
    val = 0.0; // supress annoying compiler warning
    fprintf(stderr,
        "ERROR: %s %d: proc %d: "
        "remove_based_on_binned_stat(): "
        "passed unrecognizable stat_displ %d\n",
        __FILE__, __LINE__, Myid,
        stat_displ);
    PCT_ABORT(ABORT_STATS_ERROR)
    break;
} /* end switch() */

return val;

} /* end compute_stat_val() */

static int count_histories_in_bitmask(int my_nhistories,
                                     unsigned char *history_bitmask)
{
    int n_full_bytes      = my_nhistories / 8;
    int n_remaining_bits = my_nhistories % 8;
    int i;
    int rc = 0;

    /* first counting the bits in full bytes */
    for (i = 0; i < n_full_bytes; i++)
    {
        switch (history_bitmask[i])
        {
            /* nbits = 0 */
            case 0x0: rc += 0; break;

            /* nbits = 1 */
            case 0x10: case 0x1: case 0x20: case 0x2: case 0x40: case 0x4:
            case 0x80: case 0x8:
                rc += 1; break;

            /* nbits = 2 */
            case 0x11: case 0x12: case 0x14: case 0x18: case 0x21: case 0x22:
            case 0x24: case 0x28: case 0x30: case 0x3: case 0x41: case 0x42:
            case 0x44: case 0x48: case 0x50: case 0x5: case 0x60: case 0x6:
            case 0x81: case 0x82: case 0x84: case 0x88: case 0x90: case 0x9:
            case 0xa0: case 0xa: case 0xc0: case 0xc:
                rc += 2; break;

            /* nbits = 3 */
            case 0x13: case 0x15: case 0x16: case 0x19: case 0x1a: case 0x1c:
            case 0x23: case 0x25: case 0x26: case 0x29: case 0x2a: case 0x2c:
            case 0x31: case 0x32: case 0x34: case 0x38: case 0x43: case 0x45:
            case 0x46: case 0x49: case 0x4a: case 0x4c: case 0x51: case 0x52:
            case 0x54: case 0x58: case 0x61: case 0x62: case 0x64: case 0x68:
            case 0x70: case 0x7: case 0x83: case 0x85: case 0x86: case 0x89:
            case 0x8a: case 0x8c: case 0x91: case 0x92: case 0x94: case 0x98:
            case 0xa1: case 0xa2: case 0xa4: case 0xa8: case 0xb0: case 0xb:
            case 0xc1: case 0xc2: case 0xc4: case 0xc8: case 0xd0: case 0xd:
            case 0xe0: case 0xe:
                rc += 3; break;

            /* nbits = 4 */
            case 0x17: case 0x1b: case 0x1d: case 0x1e: case 0x27: case 0x2b:
            case 0x2d: case 0x2e: case 0x33: case 0x35: case 0x36: case 0x39:
            case 0x3a: case 0x3c: case 0x47: case 0x4b: case 0x4d: case 0x4e:
            case 0x53: case 0x55: case 0x56: case 0x59: case 0x5a: case 0x5c:
            case 0x63: case 0x65: case 0x66: case 0x69: case 0x6a: case 0x6c:

```

10

20

30

40

【0 1 1 4】

【表 4 2】

```

case 0x71: case 0x72: case 0x74: case 0x78: case 0x87: case 0x8b:
case 0x8d: case 0x8e: case 0x93: case 0x95: case 0x96: case 0x99:
case 0x9a: case 0x9c: case 0xa3: case 0xa5: case 0xa6: case 0xa9:
case 0xaa: case 0xac: case 0xb1: case 0xb2: case 0xb4: case 0xb8:
case 0xc3: case 0xc5: case 0xc6: case 0xc9: case 0xca: case 0xcc:
case 0xd1: case 0xd2: case 0xd4: case 0xd8: case 0xe1: case 0xe2:
case 0xe4: case 0xe8: case 0xf0: case 0xf:
    rc += 4; break;

/* nbits = 5 */
case 0x1f: case 0x2f: case 0x37: case 0x3b: case 0x3d: case 0x3e:
case 0x4f: case 0x57: case 0x5b: case 0x5d: case 0x5e: case 0x67:
case 0x6b: case 0x6d: case 0x6e: case 0x73: case 0x75: case 0x76:
case 0x79: case 0x7a: case 0x7c: case 0x7f: case 0x8f: case 0x97: case 0x9b:
case 0x9d: case 0x9e: case 0xa7: case 0xab: case 0xad: case 0xae:
case 0xb3: case 0xb5: case 0xb6: case 0xb9: case 0xba: case 0xbc:
case 0xc7: case 0xcb: case 0xcd: case 0xce: case 0xd3: case 0xd5:
case 0xd6: case 0xd9: case 0xda: case 0xdc: case 0xe3: case 0xe5:
case 0xe6: case 0xe9: case 0xea: case 0xec: case 0xf1: case 0xf2:
case 0xf4: case 0xf8:
    rc += 5; break;

/* nbits = 6 */
case 0x3f: case 0x5f: case 0x6f: case 0x77: case 0x7b: case 0x7d:
case 0x7e: case 0x9f: case 0xaf: case 0xb7: case 0xbb: case 0xbd:
case 0xbe: case 0xcf: case 0xd7: case 0xdb: case 0xdd: case 0xde:
case 0xe7: case 0xeb: case 0xed: case 0xee: case 0xf3: case 0xf5:
case 0xf6: case 0xf9: case 0xfa: case 0xfc:
    rc += 6; break;

/* nbits = 7 */
case 0x7f: case 0xbf: case 0xdf: case 0xef: case 0xf7: case 0xfb:
case 0xfd: case 0xfe:
    rc += 7; break;

/* nbits = 8 */
case 0xff: rc += 8; break;
default:
    break;
} /* end switch() */
} /* endfor */

/* now counting the bits in last non-full byte */
for (i = 0; i < n_remaining_bits; i++)
    if (TEST(history_bitmask+n_full_bytes, i))
        rc++;

return rc;

} /* end count_histories_in_bitmask() */

static void fill_buffers_from_files(float *input_floats,
                                   unsigned short *input_ushorts,
                                   int *deg,
                                   FILE **fp,
                                   int *histories_read_from_this_file,
                                   int max_histories_per_input_file,
                                   int nhistories_buffsize,
                                   int *my_nhistories,
                                   int must_fill) // must_fill == 0 iff foreman
{
    float *float_p;
    unsigned short *ushort_p;
    int done;

    float_p = input_floats;

```

【表 4 3】

```

ushort_p = input_ushorts;

done = 0;
for (*my_nhistories = 0;
    !done && *my_nhistories < nhistories_buffsize;
    *my_nhistories = *my_nhistories + 1)
{
    if (max_histories_per_input_file
        != UNSPECIFIED_MAX_NINPUT_HISTORIES_PER_FILE
        && *histories_read_from_this_file >= max_histories_per_input_file)
    {
        /* reached limit of histories that can be read from this file */
        /* ... time to start reading from the next input file */
        fclose(*fp);

        *deg = *deg + 2;
        *fp = open_input_file(*deg);
        *histories_read_from_this_file = 0;
    } /* endif */

    if (fscanf(*fp, "%hu %hu %hu %hu %hu %hu %hu %hu %f %f %f\n",
        ushort_p+X1_DISP,  ushort_p+Y1_DISP,
        ushort_p+X2_DISP,  ushort_p+Y2_DISP,
        ushort_p+X3_DISP,  ushort_p+Y3_DISP,
        ushort_p+X4_DISP,  ushort_p+Y4_DISP,
        float_p+EIN_DISP, float_p+EOUT_DISP, float_p+THETA_DISP)
        == EOF)
    {
        /* ran out of data from open file, */
        /* closing and moving to next file */
        fclose(*fp);

        *deg = *deg + 2;
        *fp = open_input_file(*deg);
        *histories_read_from_this_file = 0;

        if (fscanf(*fp, "%hu %hu %hu %hu %hu %hu %hu %hu %f %f %f\n",
            ushort_p+X1_DISP,  ushort_p+Y1_DISP,
            ushort_p+X2_DISP,  ushort_p+Y2_DISP,
            ushort_p+X3_DISP,  ushort_p+Y3_DISP,
            ushort_p+X4_DISP,  ushort_p+Y4_DISP,
            float_p+EIN_DISP, float_p+EOUT_DISP, float_p+THETA_DISP)
            == EOF)
        {
            if (must_fill)
            {
                /* ran out of data for non-foreman workers */
                fprintf(stderr,
                    "ERROR: %s %d: proc %d: "
                    "ran out of data data deg %d\n",
                    __FILE__, __LINE__, Myid,
                    *deg);
                PCT_ABORT(ABORT_TOO_MANY_HISTORIES)
            }
            else
            {
                done = 1;
            } /* endif */
        } /* endif */
    } /* endif */

    float_p += INPUT_FLOAT_STRIDE;
    ushort_p += INPUT_USHORT_STRIDE;
    *histories_read_from_this_file = *histories_read_from_this_file + 1;
} /* endfor */

```

【表 4 4】

```

    if (done)
    {
        *my_nhistories = *my_nhistories - 1;
    } /* endif */

    return;

} /* end fill_buffers_from_files() */

static void find_entry_exit_from_vbm_or_remove (int nhistories,
                                                unsigned char *history_bitmask,
                                                float *input_floats,
                                                unsigned short *input_ushorts,
                                                struct entry_exit_t *entry_exit,
                                                int volume_bitmask_len,
                                                char *volume_bitmask)
{
    int h;
#ifdef DEBUG
    int n_cleared_histories = 0;
#endif

    for (h = 0; h < nhistories ; h++)
    {
        if (TEST(history_bitmask, h))
        {
            // intersect2.cc
            if (! compute_entry_exit_from_vbm(h,
                                                input_floats,
                                                input_ushorts,
#ifdef DEBUG
                                                volume_bitmask_len,
                                                volume_bitmask,
                                                entry_exit))
            {
                /* could NOT find entry/exit on phantom ... */
                /* declare a rogue history -- need to toss it */
                CLEAR(history_bitmask, h);
                /* NICK: need to write code to re-claim space used */
                /* NICK: in this block to record voxel ids thus */
                /* NICK: far for this history. */
#ifdef DEBUG
                n_cleared_histories ++;
#endif

                #if 0
                fprintf(stderr,
                    "ERROR: %s %d: proc %d: "
                    "computed vbm_index %d (max %d) "
                    "from history %d "
                    "x1 %d y1 %d x2 %d y2 %d x3 %d y3 %d x4 %d y4 %d "
                    "entry_x %f entry_y %f entry_z %f "
                    "exit_x %f exit_y %f exit_z %f "
                    "\n",
                    __FILE__, __LINE__, Myid,
                    voxel_idx, volume_bitmask_len,
                    h,
                    get_x1(input_ushorts, h), get_y1(input_ushorts, h),
                    get_x2(input_ushorts, h), get_y2(input_ushorts, h),
                    get_x3(input_ushorts, h), get_y3(input_ushorts, h),
                    get_x4(input_ushorts, h), get_y4(input_ushorts, h),
                    entry_exit.entry_x, entry_exit.entry_y, entry_exit.entry_z,
                    entry_exit.exit_x, entry_exit.exit_y, entry_exit.exit_z);
                #endif
            }
        }
    }
}

```

【表 4 5】

```

#endif
/* MPI::COMM_WORLD.Abort(ABORT_INVALID_IDX); */
#endif

    } /* endif */
} /* endif */
} /* endfor */

#ifdef DEBUG
    printf("%d: could not connect to phantom n_cleared_histories %d\n",
        Myid, n_cleared_histories);
#endif

    return;

} /* end find_entry_exit_from_vbm_or_remove() */

static int get_stat_cell_idx(float *input_floats,
                            unsigned short *input_ushorts,
                            int h,
                            int stat_displ)
{
    /* binning based on INTERNAL ENTRY plane */
    int x = get_x2(input_ushorts, h);
    int y = get_y2(input_ushorts, h);

    int converted_x = (x * REGION_COLS)/MAX_TRACK_X_IDX;
    int converted_y = (y * REGION_ROWS)/MAX_TRACK_Y_IDX;

    float theta = get_theta(input_floats, h);
    int theta_idx = (int)(theta * THETA_IDX_PER_DEGREE + 0.5);

#ifdef DEBUG
    {
        int rc = ( (STAT_CELL_IDX_ANGLE_STRIDE * theta_idx)
                  + (STAT_CELL_IDX_ROW_STRIDE * converted_y)
                  + (STAT_CELL_IDX_COL_STRIDE * converted_x)
                  + stat_displ);

        if (rc < 0 || rc >= N_STAT_CELLS)
        {
            fprintf(stderr,
                "ERROR: %s %d: proc %d: "
                "computed invalid stats_idx %d "
                "from history %d x %d y %d theta %f stat_displ %d "
                "(must be 0 <= stats_idx < N_STAT_CELLS %d)\n",
                __FILE__, __LINE__, Myid,
                rc, h, x, y, theta, stat_displ, N_STAT_CELLS);
            PCT_ABORT(ABORT_STATS_ERROR)
        } /* endif */

        return rc;
    }
#else
    return ( (STAT_CELL_IDX_ANGLE_STRIDE * theta_idx)
            + (STAT_CELL_IDX_ROW_STRIDE * converted_y)
            + (STAT_CELL_IDX_COL_STRIDE * converted_x)
            + stat_displ);
#endif

} /* end get_stat_cell_idx() */

static FILE *open_input_file(int degree)
{

```

10

20

30

40

【表 4 6】

```

FILE *fp;
char inputfilename[256];

sprintf(inputfilename, "%s/%s%03d%s",
        InputDir, InputFnamePrefix, degree, InputFnameSuffix);

if (!(fp = fopen(inputfilename, "r")))
{
    fprintf(stderr, "ERROR: could not open input file >%s<\n",
            inputfilename);
    PCT_ABORT(ABORT_FAILED_FOPEN)
} /* endif */

return fp;

} /* end open_input_file() */

/* extern "C" int getopt(int argc, char * const argv[], const char *optstring) throw();
*/
extern "C" int getopt(int argc, char * const argv[], const char *optstring);

static void parse_command_line_args(int argc,
                                    char **argv,
                                    int *n_input_histories,
                                    int *max_histories_per_foreman_or_worker,
                                    int *max_histories_per_input_file,
                                    float *linear_solver_relaxation)
{
    int c;
    enum input_sets {threeM, thirtyeightM, twoB} input;
    int error;

    /* int getopt(int argc, char * const argv[], const char *optstring); */
    extern char *optarg;
    /* extern int optind, opterr, optopt; */

    error = 0;

    /* default values */
    *n_input_histories = UNSPECIFIED_NINPUT_HISTORIES;
    *max_histories_per_input_file = UNSPECIFIED_MAX_NINPUT_HISTORIES_PER_FILE;
    *max_histories_per_foreman_or_worker = DEFAULT_MAXHISTORIES_PER_WORKER;
    input = thirtyeightM;
    *linear_solver_relaxation = DEFAULT_RELAXATION;

    while ((c = getopt(argc, argv, "sblh:f:m:r:")) != -1)
    {
        switch (c)
        {
            case 's': input = threeM; break;
            case 'b': input = thirtyeightM; break;
            case 'l': input = twoB; break;
            case 'h': *n_input_histories = atoi(optarg); break;
            case 'f': *max_histories_per_input_file=atoi(optarg); break;
            case 'm': *max_histories_per_foreman_or_worker=atoi(optarg); break;
            case 'r': *linear_solver_relaxation=atof(optarg); break;
            default: error = 1; break;
        } /* end switch() */
    } /* endwhile */

    if (error)
    {
        if (Myid == ForemanRank)
        {

```

10

20

30

40

【表 4 7】

```

/* only Master prints usage message */
fprintf(stderr,
    "\n\tusage: %s {-s | -b | -l} "
    "{-h <N input histories>} "
    "{-f <max N input histories per file>} "
    "{-m <max N histories per worker or foreman>} "
    "{-r <linear solver relaxation param>} "
    "\n\n",
    argv[0]);
fprintf(stderr, "where\n\n");
fprintf(stderr, "\t-s - small input, 2D, ~3M histories\n");
fprintf(stderr, "\t-b - big input, 3D, ~38M histories (default)\n");
fprintf(stderr, "\t-l - large input, 3D, ~1.9B histories\n");
fprintf(stderr, "\n");

#ifdef NO_MPI
    exit(1);
#else
    MPI_Abort(MPI_COMM_WORLD, ABORT_INVALID_CMD_LINE_ARGS);
#endif
} /* endif */
} /* endif */

switch (input)
{
    case threeM:
        sprintf(InputDir, "%s/Input-2D-3.2M/", InputDirBase);
        strcpy(InputFnamePrefix, "projections.");
        strcpy(InputFnameSuffix, ".dat");
        break;

    case thirtyeightM:
        sprintf(InputDir, "%s/Input-Full3D-38.6M/", InputDirBase);
        strcpy(InputFnamePrefix, "angle_");
        strcpy(InputFnameSuffix, ".txt");
        break;

    case twoB:
        sprintf(InputDir, "%s/Input-Full3D-GeneratedAtNIU-1.9B/ConvertedFromPen
fold/",
                InputDirBase);
        strcpy(InputFnamePrefix, "angle_");
        strcpy(InputFnameSuffix, ".txt");
        break;

    default:
        fprintf(stderr,
            "ERROR: parse_command_line_args(): "
            "encountered invalid input = %d\n",
            input);

#ifdef NO_MPI
    exit(1);
#else
    MPI_Abort(MPI_COMM_WORLD, ABORT_INVALID_CMD_LINE_ARGS);
#endif
    break;

} /* end switch */

return;

} /* end parse_command_line_args() */

static void read_and_distribute_input(int histories_per_worker,
                                     int max_histories_per_input_file,

```

【表 4 8】

```

int *my_nhistories,
int must_fill_foreman,
float *input_floats,
unsigned short *input_ushorts)

{
    int deg;
    int histories_read_from_this_file;
#ifdef NO_MPI
    int i;
    MPI_Status status;
#endif
    FILE *fp;

    /* NICK: fix this - set to received for worker or last read for foreman*/

    /* max_histories_per_foreman_or_worker is always defined */
    /* in that context there are two cases: */
    /* (1) we don't know how many input histories there are: */
    /*     read max_histories_per_foreman_or_worker into */
    /*     each process, must fill all workers, foreman is */
    /*     last to get his and so may be less than full */
    /* (2) we know how many input histories there are: */
    /*     spread histories evenly across all Nprocs, make */
    /*     sure under max_histories_per_foreman_or_worker, */
    /*     all workers get the same number of histories and */
    /*     foreman may get more due to rounding */

    /******
    /* Read and distribute input */
    /******

    if (Myid == ForemanRank)
    {
        /* I am the foreman */

        TotalHistories = 0;

        /* opening first file */
        deg = 0;
        fp = open_input_file(deg);
        histories_read_from_this_file = 0;;

#ifdef NO_MPI
        /* must read and distribute input to other workers */
        for (i = 1; i < Nprocs; i++)
        {
            fill_buffers_from_files(input_floats,
                                   input_ushorts,
                                   &deg,
                                   &fp,
                                   &histories_read_from_this_file,
                                   max_histories_per_input_file,
                                   histories_per_worker,
                                   my_nhistories,
                                   1); // must_fill

            MPI_Send(input_floats,
                    histories_per_worker*INPUT_FLOAT_STRIDE,
                    MPI_FLOAT,
                    i,
                    INPUT_FLOAT_TAG,
                    MPI_COMM_WORLD);
            MPI_Send(input_ushorts,
                    histories_per_worker*INPUT_USHORT_STRIDE,
                    MPI_UNSIGNED_SHORT,
                    i,
                    INPUT_USHORT_TAG,
                    MPI_COMM_WORLD);
        }

```

10

20

30

40

【表 4 9】

```

        i,                                /* dest */
        INPUT_USHORT_TAG,                 /* tag */
        MPI_COMM_WORLD);                 /* comm */

    TotalHistories += histories_per_worker;

} /* endfor */
#endif

/* now fill foreman's buffer, which may not be full */
fill_buffers_from_files(input_floats,
                        input_ushorts,
                        &deg,
                        &fp,
                        &histories_read_from_this_file,
                        max_histories_per_input_file,
                        *my_nhistories,
                        my_nhistories,    // may reset foreman's val
                        must_fill_foreman); // must_fill

    TotalHistories += (*my_nhistories);
}
#ifdef NO_MPI
else
{
    /* I am NOT the foreman */

    /* must receive my input from the foreman */
    MPI_Recv(input_floats,
             histories_per_worker*INPUT_FLOAT_STRIDE,
             MPI_FLOAT,
             ForemanRank,
             INPUT_FLOAT_TAG,
             MPI_COMM_WORLD,
             &status);
    MPI_Recv(input_ushorts,
             histories_per_worker*INPUT_USHORT_STRIDE,
             MPI_UNSIGNED_SHORT,
             ForemanRank,
             INPUT_USHORT_TAG,
             MPI_COMM_WORLD,
             &status);

} /* endif */
#endif

return;

} /* end read_and_distribute_input() */

static void remove_based_on_binned_stat(int nhistories,
                                       unsigned char *history_bitmask,
                                       float *input_floats,
                                       unsigned short *input_ushorts,
                                       struct stats s[],
                                       float sigma_cutoff[],
                                       int stat_displ)
{
    int h;
    int stat_idx;
    float val;
#ifdef DEBUG
    int n_cleared_histories = 0;
#endif

    for (h = 0; h < nhistories; h++)

```

【表 5 0】

```

{
    if (TEST(history_bitmask, h))
    {
        /* this history has not been previously thrown out */

        stat_idx = get_stat_cell_idx(input_floats,
                                    input_ushorts,
                                    h,
                                    stat_displ);

#ifdef DEBUG
        /* sanity check */
        /* confirming that history has not yet been cleared */
        /* has been accumulated as part of the global stats */

        if (s[stat_idx].n <= 0)
        {
            fprintf(stderr,
                "ERROR: %s %d: proc %d: "
                "history %d: stat_idx %d from stat_displ %d "
                "found n = %d (must be > 0)\n",
                __FILE__, __LINE__, Myid,
                h, stat_idx, stat_displ, s[stat_idx].n);
            PCT_ABORT(ABORT_STATS_ERROR)
        } /* endif */
#endif

        val = compute_stat_val(input_floats,
                               input_ushorts,
                               h,
                               stat_displ);

#ifdef DEBUG
        /* sanity check */
        /* confirming global_min_x <= x <= global_max_x
        /* where "global" means across all procs for stats cell stat_idx */

        if (val < s[stat_idx].min)
        {
            fprintf(stderr,
                "ERROR: %s %d: proc %d: "
                "history %d: stat_idx %d from stat_displ %d "
                "computed val %f < global min %f\n",
                __FILE__, __LINE__, Myid,
                h, stat_idx, stat_displ, val, s[stat_idx].min);
            PCT_ABORT(ABORT_STATS_ERROR)
        } /* endif */

        if (val > s[stat_idx].max)
        {
            fprintf(stderr,
                "ERROR: %s %d: proc %d: "
                "history %d: stat_idx %d from stat_displ %d "
                "computed val %f > global max %f\n",
                __FILE__, __LINE__, Myid,
                h, stat_idx, stat_displ, val, s[stat_idx].max);
            PCT_ABORT(ABORT_STATS_ERROR)
        }
    }
}

```

【 0 1 2 3】

【表 5 1】

```

        } /* endif */
#endif

        if (fabs(val - s[stat_idx].mean) > sigma_cutoff[stat_idx])
        {
            CLEAR(history_bitmask, h);
#ifdef DEBUG
            n_cleared_histories ++;
#endif
        } /* endfor */
    } /* endif */
} /* endfor */

#ifdef DEBUG
    printf("%d: stat_displ %d n_cleared_histories %d\n",
        Myid, stat_displ, n_cleared_histories);
#endif

    return;

} /* end remove_based_on_binned_stat() */

static void remove_histories_from_binned_stats(int nhistories,
                                                unsigned char *history_bitmask,
                                                float *input_floats,
                                                unsigned short *input_ushorts)
{
    int h;
    int stat_idx;
    float val;
    /* could make s[] and sigma_cutoff[] local vars instead */
    /* of malloc/free them, but their combined size alone is ~11MB */
    /* which on some systems blows the default stack size (e.g., 8MB) */
    /* ... easier to malloc/free them rather than wrestle with stack sizes */
    /* struct stats s[N_STAT_CELLS]; */
    /* float sigma_cutoff[N_STAT_CELLS]; */
    struct stats *s;
    float *sigma_cutoff;

    PCT_MALLOC(s, N_STAT_CELLS*sizeof(struct stats), struct stats);
    PCT_MALLOC(sigma_cutoff, N_STAT_CELLS*sizeof(float), float);

    /* printf("%d: ENTER remove_histories_from_binned_stats()\n", Myid); */
    /* initializing stats */
    for (h = 0; h < N_STAT_CELLS; h++)
        s[h].n = 0;

    /* ***** */
    /* PHASE 1 of 3 - compute local running mean and SSE */
    /* ***** */

    for (h = 0; h < nhistories; h++)
    {
        if (TEST(history_bitmask, h))
        {
            /* ***** */
            /* accumulating stats for delta x */
            /* ***** */

            stat_idx = get_stat_cell_idx(input_floats,
                                        input_ushorts,
                                        h,
                                        STAT_DELTA_X);

```

10

20

30

40

【 0 1 2 4】

50

【表 5 2】

```

        val = compute_stat_val(input_floats,
                                input_ushorts,
                                h,
                                STAT_DELTA_X);

        accumulate_stats_val(&(s[stat_idx].mean),
                              &(s[stat_idx].sse),
#ifdef DEBUG
                              &(s[stat_idx].min),
                              &(s[stat_idx].max),
#elseif
                              &(s[stat_idx].n),
                              val);

        /* *****
        /* accumulating stats for delta y */
        /* *****

        stat_idx = get_stat_cell_idx(input_floats,
                                      input_ushorts,
                                      h,
                                      STAT_DELTA_Y);

        val = compute_stat_val(input_floats,
                                input_ushorts,
                                h,
                                STAT_DELTA_Y);

        accumulate_stats_val(&(s[stat_idx].mean),
                              &(s[stat_idx].sse),
#ifdef DEBUG
                              &(s[stat_idx].min),
                              &(s[stat_idx].max),
#elseif
                              &(s[stat_idx].n),
                              val);

        /* *****
        /* accumulating stats for delta energy */
        /* *****

        stat_idx = get_stat_cell_idx(input_floats,
                                      input_ushorts,
                                      h,
                                      STAT_DELTA_ENERGY);

        val = compute_stat_val(input_floats,
                                input_ushorts,
                                h,
                                STAT_DELTA_ENERGY);

        accumulate_stats_val(&(s[stat_idx].mean),
                              &(s[stat_idx].sse),
#ifdef DEBUG
                              &(s[stat_idx].min),
                              &(s[stat_idx].max),
#elseif
                              &(s[stat_idx].n),
                              val);

        } /* endif */
    } /* endfor */

#ifdef DEBUG
    /* *****

```

【 0 1 2 5】

【表 5 3】

```

/* sanity check */
/*****
{
    int i,j;

    for (i = 0; i < N_STAT_CELLS; i += STAT_NSTATS)
    {
        for (j = 1; j < STAT_NSTATS; j ++)
        {
            if (s[i].n != s[i+j].n)
            {
                fprintf(stderr,
                    "ERROR: %s %d: proc %d: "
                    "LOCAL STAT: stats_idx %d: "
                    "found n_0 = %d != n_%d = %d "
                    "(must be equal)\n",
                    FILE_, LINE_, Myid,
                    h, s[i].n, j, s[i+j].n);
                PCT_ABORT(ABORT_STATS_ERROR)
            } /* endif */
        } /* endfor */
    } /* endfor */

    for (i = 0; i < N_STAT_CELLS; i += STAT_NSTATS)
    {
        for (j = 0; j < STAT_NSTATS; j ++)
        {
            if (s[i+j].n > 2 && s[i+j].sse <= 0.0)
            {
                fprintf(stderr,
                    "WARNING: %s %d: proc %d: "
                    "LOCAL STAT: stats_idx %d: found n_%d = %d sse_%d = %f "
                    "(sse probably should be > 0)\n",
                    FILE_, LINE_, Myid,
                    i, j, s[i+j].n, j, s[i+j].sse);
            } /* endif */
        } /* endfor */
    } /* endfor */
}
#endif

/*****
/* PHASE 2 of 3 - compute and distribute global mean and SSE */
/*****
#ifdef NO_MPI
{
    MPI_Datatype Stats_mpi;
#ifdef DEBUG
    int block_lengths[5]; /* number of "things" in each element of */
                        /* our new struct. for us, 1 each */
    MPI_Aint displs[5]; /* displacement of each element */
    MPI_Datatype types[5]; /* type of each element */
#else
    int block_lengths[3]; /* number of "things" in each element of */
                        /* our new struct. for us, 1 each */
    MPI_Aint displs[3]; /* displacement of each element */
    MPI_Datatype types[3]; /* type of each element */
#endif
    MPI_Aint start_addr;
    MPI_Aint addr;
    MPI_Op op;

    block_lengths[0] = block_lengths[1] = block_lengths[2]
#ifdef DEBUG
    = block_lengths[3] = block_lengths[4]

```

【表 5 4】

```

-
#endif
    = 1;

    types[0] = MPI_FLOAT;
    types[1] = MPI_FLOAT;
    types[2] = MPI_UNSIGNED;
#ifdef DEBUG
    types[3] = MPI_FLOAT;
    types[4] = MPI_FLOAT;
#endif

    MPI_Address(&(s[0].mean), &start_addr);
    displs[0] = 0;

    MPI_Address(&(s[0].sse), &addr);
    displs[1] = addr - start_addr;

    MPI_Address(&(s[0].n), &addr);
    displs[2] = addr - start_addr;

#ifdef DEBUG
    MPI_Address(&(s[0].min), &addr);
    displs[3] = addr - start_addr;

    MPI_Address(&(s[0].max), &addr);
    displs[4] = addr - start_addr;
#endif

    /* build new derived datatype */
    MPI_Type_struct(
#ifdef DEBUG
        5, /* nelements */
#else
        3, /* nelements */
#endif
        block_lengths,
        displs,
        types,
        &Stats_mpi);
    MPI_Type_commit(&Stats_mpi);

    MPI_Op_create((MPI_User_function*) combine_stats, // my user-def func
        1, // commute flag
        &op); // op handle

    MPI_Allreduce(MPI_IN_PLACE, // clobber operand with result
        s, // operand that gets clobbered by result
        N_STAT_CELLS, // count
        Stats_mpi, // data type
        op, // operator
        MPI_COMM_WORLD);

    MPI_Op_free(&op);
}
#endif

#ifdef DEBUG
    /* sanity check */
    {
        int i,j;

        for (i = 0; i < N_STAT_CELLS; i += STAT_NSTATS)
        {

```

【表 5 5】

```

    for (j = 1; j < STAT_NSTATS; j++)
    {
        if (s[i].n != s[i+j].n)
        {
            fprintf(stderr,
                "ERROR: %s %d: proc %d: "
                "GLOBAL STAT: stats_idx %d: "
                "found n_0 = %d != n_%d = %d "
                "(must be equal)\n",
                FILE_, LINE_, Myid,
                h, s[i].n, j, s[i+j].n);
            PCT_ABORT(ABORT_STATS_ERROR)
        } /* endif */
    } /* endfor */
} /* endfor */

for (i = 0; i < N_STAT_CELLS; i += STAT_NSTATS)
{
    for (j = 0; j < STAT_NSTATS; j++)
    {
        if (s[i+j].n > 2 && s[i+j].sse <= 0.0)
        {
            fprintf(stderr,
                "WARNING: %s %d: proc %d: "
                "GLOBAL STAT: "
                "stats_idx %d: found n_%d = %d sse_%d = %f "
                "(sse probably should be > 0)\n",
                FILE_, LINE_, Myid,
                i, j, s[i+j].n, j, s[i+j].sse);
        } /* endif */
    } /* endfor */
} /* endfor */
}
#endif

/* computing sigma cutoff for each stat cell */
for (h = 0; h < N_STAT_CELLS; h++)
{
    if (s[h].n == 0)
        sigma_cutoff[h] = 0; // arbitrary value, should never be used
    else
        //KIRK:sigma_cutoff[h]=s[h].mean+(MAX_SIGMA*sqrt(s[h].sse/(s[h].n)));
        sigma_cutoff[h] = MAX_SIGMA*sqrt(s[h].sse/s[h].n);
} /* endfor */

/* dumping stats */
{
    FILE *fp_x, *fp_y, *fp_e;
    FILE *fp = NULL;
    char fname[300];

    if (Myid == ForemanRank)
    {
        sprintf(fname, "%s/stats.x.%s.%s",
            OutputDirBase, Nowstring, Relaxparamstring);
        if ((fp_x = fopen(fname, "w")) == NULL)
        {
            fprintf(stderr, "ERROR: could not open output stats file >%s<\n",
                fname);
            PCT_ABORT(ABORT_FAILED_FOPEN)
        } /* endif */

        sprintf(fname, "%s/stats.y.%s.%s",
            OutputDirBase, Nowstring, Relaxparamstring);
        if ((fp_y = fopen(fname, "w")) == NULL)

```

【0 1 2 8】

【表 5 6】

```

{
    fprintf(stderr, "ERROR: could not open output stats file >%s<\n",
               fname);
    PCT_ABORT(ABORT_FAILED_FOPEN)
} /* endif */

sprintf(fname, "%s/stats.e.%s.%s",
        OutputDirBase, Nowstring, Relaxparamstring);
if ((fp_e = fopen(fname, "w")) == NULL)
{
    fprintf(stderr, "ERROR: could not open output stats file >%s<\n",
               fname);
    PCT_ABORT(ABORT_FAILED_FOPEN)
} /* endif */

for (h = 0; h < N_STAT_CELLS; h++)
{
    if ((h % 3) == 0) fp = fp_x;
    if ((h % 3) == 1) fp = fp_y;
    if ((h % 3) == 2) fp = fp_e;
#ifdef DEBUG
    fprintf(fp, "%d %f %f %f %f %f\n",
            s[h].n, s[h].mean, s[h].sse, sigma_cutoff[h],
            s[h].min, s[h].max);
#else
    fprintf(fp, "%d %f %f %f\n",
            s[h].n, s[h].mean, s[h].sse, sigma_cutoff[h]);
#endif
} /* endfor */

fclose(fp_x);
fclose(fp_y);
fclose(fp_e);
} /* endif */

}

/*****
/* PHASE 3 of 3 - throwing out rogue histories (outside MAX_SIGMA) */
*****/

remove_based_on_binned_stat(nhistories,
                             history_bitmask,
                             input_floats,
                             input_ushorts,
                             s,
                             sigma_cutoff,
                             STAT_DELTA_X);

remove_based_on_binned_stat(nhistories,
                             history_bitmask,
                             input_floats,
                             input_ushorts,
                             s,
                             sigma_cutoff,
                             STAT_DELTA_Y);

remove_based_on_binned_stat(nhistories,
                             history_bitmask,
                             input_floats,
                             input_ushorts,
                             s,
                             sigma_cutoff,
                             STAT_DELTA_ENERGY);

```

【0 1 2 9】

【表 5 7】

```

    PCT_FREE(s);
    PCT_FREE(sigma_cutoff);

    return;

} /* end remove_histories_from_binned_stats() */

// NICK: need to write test_solution_for_done()
static int test_solution_for_done(float solution[])
{
    int rc;

    Iterations ++;
    rc = (Iterations >= Niters);

    return rc;
} /* end test_solution() */
#include "pct.h"

#ifdef NO_MPI

int MPI_Init(int *argc, char ***argv)
{
    int idx = MPI_INIT_IDX;
    int rc;
    int i;
    double start_time, end_time;

    start_time = MPI_Wtime();

    strcpy(func_profiles[MPI_INIT_IDX].func_name, "MPI_Init");
    strcpy(func_profiles[MPI_COMM_SIZE_IDX].func_name, "MPI_Comm_size");
    strcpy(func_profiles[MPI_COMM_RANK_IDX].func_name, "MPI_Comm_rank");
    strcpy(func_profiles[MPI_ADDRESS_IDX].func_name, "MPI_Address");
    strcpy(func_profiles[MPI_TYPE_STRUCT_IDX].func_name, "MPI_Type_struct");
    strcpy(func_profiles[MPI_TYPE_COMMIT_IDX].func_name, "MPI_Type_commit");
    strcpy(func_profiles[MPI_OP_CREATE_IDX].func_name, "MPI_Op_create");
    strcpy(func_profiles[MPI_OP_FREE_IDX].func_name, "MPI_Op_free");
    strcpy(func_profiles[MPI_ALLREDUCE_IDX].func_name, "MPI_Allreduce");
    strcpy(func_profiles[MPI_BCAST_IDX].func_name, "MPI_Bcast");
    strcpy(func_profiles[MPI_SEND_IDX].func_name, "MPI_Send");
    strcpy(func_profiles[MPI_RECV_IDX].func_name, "MPI_Recv");

    for (i = 0; i < Nfuncs; i++)
    {
        func_profiles[i].call_count = 0;
        func_profiles[i].total_time_secs = 0.0;
        func_profiles[i].does_communicate = 0;
        func_profiles[i].total_bytes = 0;
    } /* endfor */

    /* fixing those that DO communicate */
    func_profiles[MPI_ALLREDUCE_IDX].does_communicate
        = func_profiles[MPI_BCAST_IDX].does_communicate
        = func_profiles[MPI_SEND_IDX].does_communicate
        = func_profiles[MPI_RECV_IDX].does_communicate = 1;

    rc = PMPI_Init(argc, argv);

    end_time = MPI_Wtime();

    (func_profiles[idx].call_count) ++;

```

【 0 1 3 0】

【表 5 8】

```

        (func_profiles[idx].total_time_secs) += (end_time - start_time);

        return rc;

    } /* end MPI_Init() */

int MPI_Comm_size(MPI_Comm comm, int *size)
{
    int idx = MPI_COMM_SIZE_IDX;
    int rc;
    double start_time, end_time;

    start_time = MPI_Wtime();
    rc = PMPI_Comm_size(comm, size);
    end_time = MPI_Wtime();

    (func_profiles[idx].call_count) ++;
    (func_profiles[idx].total_time_secs) += (end_time - start_time);

    return rc;

} /* end MPI_Comm_size() */

int MPI_Comm_rank(MPI_Comm comm, int *rank)
{
    int idx = MPI_COMM_RANK_IDX;
    int rc;
    double start_time, end_time;

    start_time = MPI_Wtime();
    rc = PMPI_Comm_rank(comm, rank);
    end_time = MPI_Wtime();

    (func_profiles[idx].call_count) ++;
    (func_profiles[idx].total_time_secs) += (end_time - start_time);

    return rc;

} /* end MPI_Comm_rank() */

int MPI_Address(void *location, MPI_Aint *address)
{
    int idx = MPI_ADDRESS_IDX;
    int rc;
    double start_time, end_time;

    start_time = MPI_Wtime();
    rc = PMPI_Address(location, address);
    end_time = MPI_Wtime();

    (func_profiles[idx].call_count) ++;
    (func_profiles[idx].total_time_secs) += (end_time - start_time);

    return rc;

} /* end MPI_Address() */

int MPI_Type_struct(int count,
                    int *array_of_blocklengths,
                    MPI_Aint *array_of_displacements,
                    MPI_Datatype *array_of_types,
                    MPI_Datatype *newtype)
{
    int idx = MPI_TYPE_STRUCT_IDX;
    int rc;

```

【表 5 9】

```

double start_time, end_time;

start_time = MPI_Wtime();
rc = PMPI_Type_struct(count,
                      array_of_blocklengths,
                      array_of_displacements,
                      array_of_types,
                      newtype);
end_time = MPI_Wtime();

(func_profiles[idx].call_count) ++;
(func_profiles[idx].total_time_secs) += (end_time - start_time);

return rc;

} /* end MPI_Type_struct() */

int MPI_Type_commit(MPI_Datatype *datatype)
{
    int idx = MPI_TYPE_COMMIT_IDX;
    int rc;
    double start_time, end_time;

    start_time = MPI_Wtime();
    rc = PMPI_Type_commit(datatype);
    end_time = MPI_Wtime();

    (func_profiles[idx].call_count) ++;
    (func_profiles[idx].total_time_secs) += (end_time - start_time);

    return rc;

} /* end MPI_Type_commit() */

int MPI_Op_create(MPI_User_function *function, int commute, MPI_Op *op)
{
    int idx = MPI_OP_CREATE_IDX;
    int rc;
    double start_time, end_time;

    start_time = MPI_Wtime();
    rc = PMPI_Op_create(function, commute, op);
    end_time = MPI_Wtime();

    (func_profiles[idx].call_count) ++;
    (func_profiles[idx].total_time_secs) += (end_time - start_time);

    return rc;

} /* end MPI_Op_create() */

int MPI_Op_free(MPI_Op *op)
{
    int idx = MPI_OP_FREE_IDX;
    int rc;
    double start_time, end_time;

    start_time = MPI_Wtime();
    rc = PMPI_Op_free(op);
    end_time = MPI_Wtime();

    (func_profiles[idx].call_count) ++;
    (func_profiles[idx].total_time_secs) += (end_time - start_time);

    return rc;

```

【表 6 0】

```

} /* end MPI_Op_free() */

/*****
/* Message Passing functions */
*****/

int MPI_Allreduce(void *sendbuf,
                  void *recvbuf,
                  int count,
                  MPI_Datatype datatype,
                  MPI_Op op,
                  MPI_Comm comm)
{
    int idx = MPI_ALLREDUCE_IDX;
    int rc;
    double start_time, end_time;
    int extent;

    start_time = MPI_Wtime();
    rc = PMPI_Allreduce(sendbuf, recvbuf, count, datatype, op, comm);
    end_time = MPI_Wtime();

    (func_profiles[idx].call_count) ++;
    (func_profiles[idx].total_time_secs) += (end_time - start_time);

    MPI_Type_size(datatype, &extent);
    (func_profiles[idx].total_bytes) += (count * extent);

    return rc;
} /* end MPI_Allreduce() */

int MPI_Bcast(void *buffer,
              int count,
              MPI_Datatype datatype,
              int root,
              MPI_Comm comm)
{
    int idx = MPI_BCAST_IDX;
    int rc;
    double start_time, end_time;
    int extent;

    start_time = MPI_Wtime();
    rc = PMPI_Bcast(buffer, count, datatype, root, comm);
    end_time = MPI_Wtime();

    (func_profiles[idx].call_count) ++;
    (func_profiles[idx].total_time_secs) += (end_time - start_time);

    MPI_Type_size(datatype, &extent);
    (func_profiles[idx].total_bytes) += (count * extent);

    return rc;
} /* end MPI_Bcast() */

int MPI_Send(void *buf,
             int count,
             MPI_Datatype datatype,
             int dest,
             int tag,
             MPI_Comm comm)
{

```

【 0 1 3 3 】

【表 6 1】

```

-
int idx = MPI_SEND_IDX;
int rc;
double start_time, end_time;
int extent;

start_time = MPI_Wtime();
rc = PMPI_Send(buf, count, datatype, dest, tag, comm);
end_time = MPI_Wtime();

(func_profiles[idx].call_count) ++;
(func_profiles[idx].total_time_secs) += (end_time - start_time);

MPI_Type_size(datatype, &extent);
(func_profiles[idx].total_bytes) += (count * extent);

return rc;
} /* end MPI_Send() */

int MPI_Recv(void *buf,
             int count,
             MPI_Datatype datatype,
             int source,
             int tag,
             MPI_Comm comm,
             MPI_Status *status)
{
    int idx = MPI_RECV_IDX;
    int rc;
    double start_time, end_time;
    int extent;

    start_time = MPI_Wtime();
    rc = PMPI_Recv(buf, count, datatype, source, tag, comm, status);
    end_time = MPI_Wtime();

    (func_profiles[idx].call_count) ++;
    (func_profiles[idx].total_time_secs) += (end_time - start_time);

    MPI_Type_size(datatype, &extent);
    (func_profiles[idx].total_bytes) += (count * extent);

    return rc;
} /* end MPI_Recv() */

#endif

```

10

20

30

40

番号別に参照された。刊行物の全体引用は、以下に列挙される。これらの刊行物および特許の開示は、その全体において、本発明が関わる最新技術をより完全に説明するために、参照することによって本願に組み込まれる。

【0134】

本発明は、例示的に説明されており、使用された専門用語は、限定ではなく、説明の用語の性質におけるものであることが意図されることを理解されたい。

【0135】

明らかに、本発明の多くの修正および変形例が、前述の教示に照らして可能である。したがって、添付の請求項の範囲内において、本発明は、具体的に説明される以外にも実践することができることを理解されたい。

【図2】

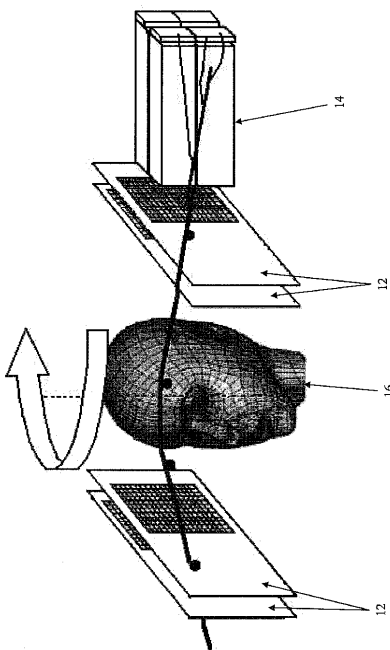


FIGURE 2

【図3】

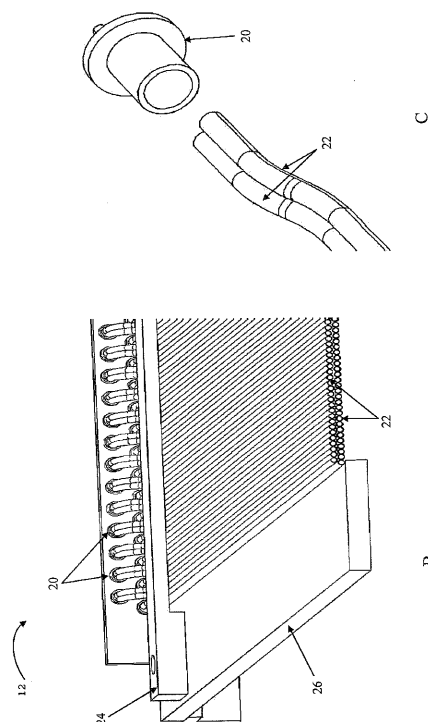


FIGURE 3

【図 3 A】

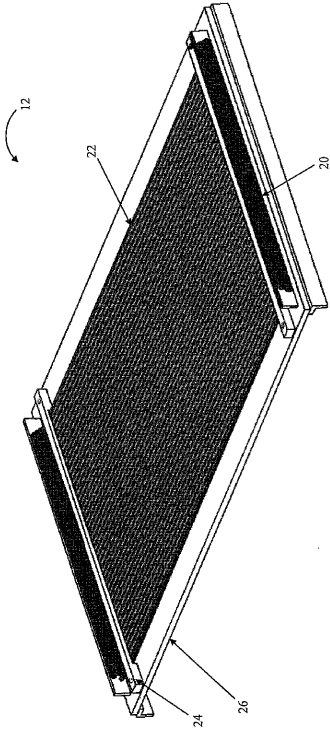


FIGURE 3A

【図 4】

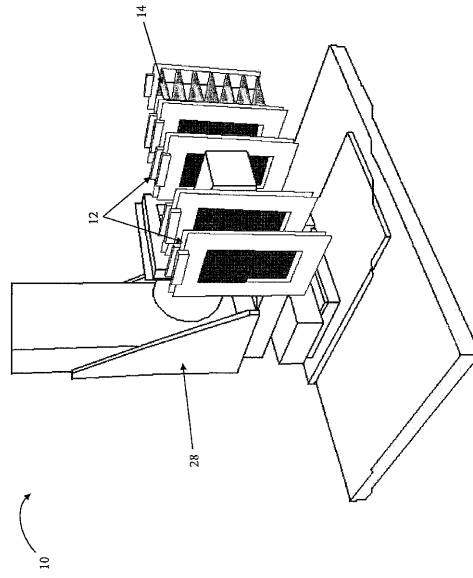


FIGURE 4

【図 7】

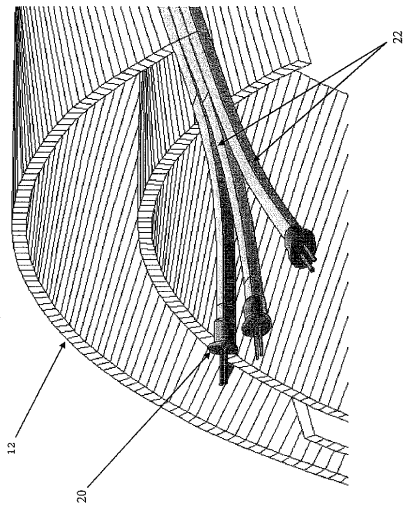


FIGURE 7

【図 8】

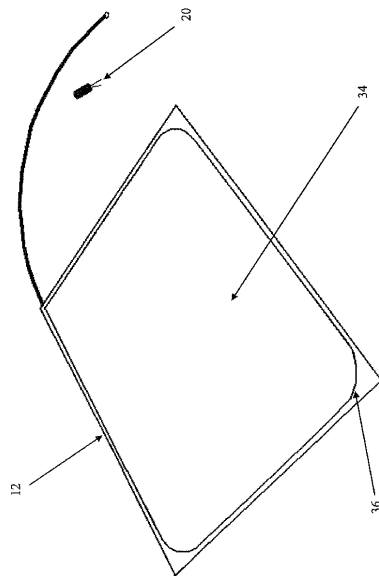


FIGURE 8

【図 1】

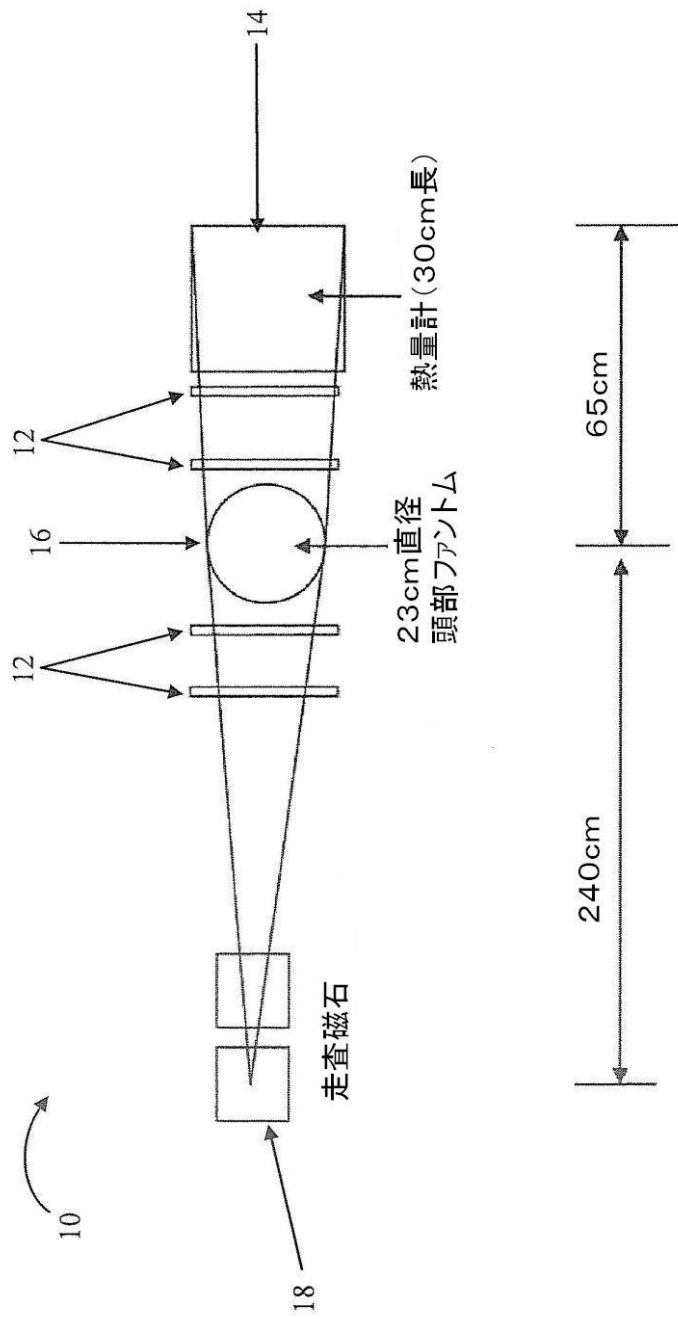


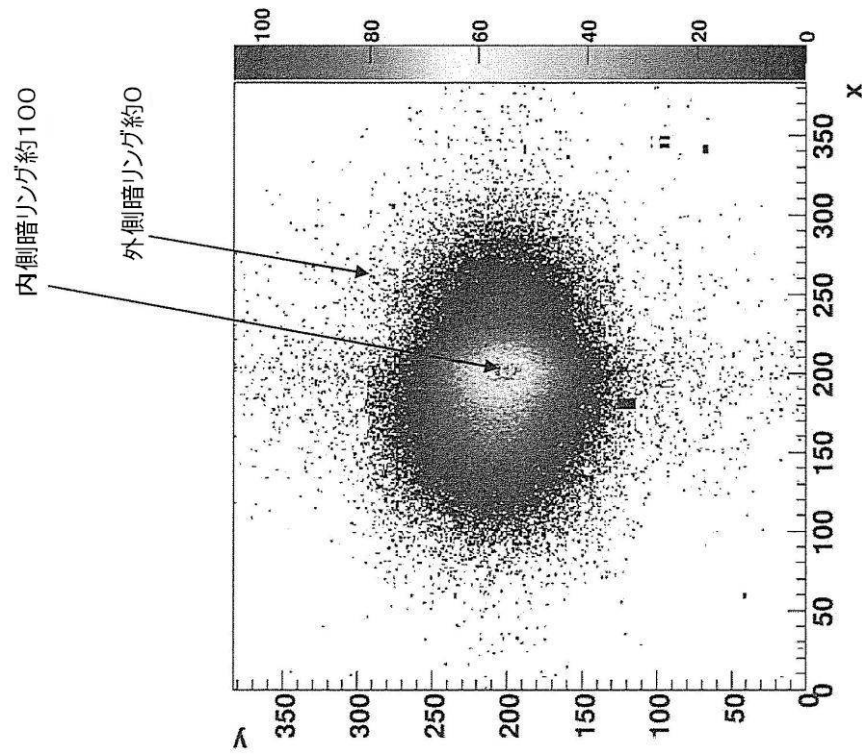
FIGURE 1

【図 5】

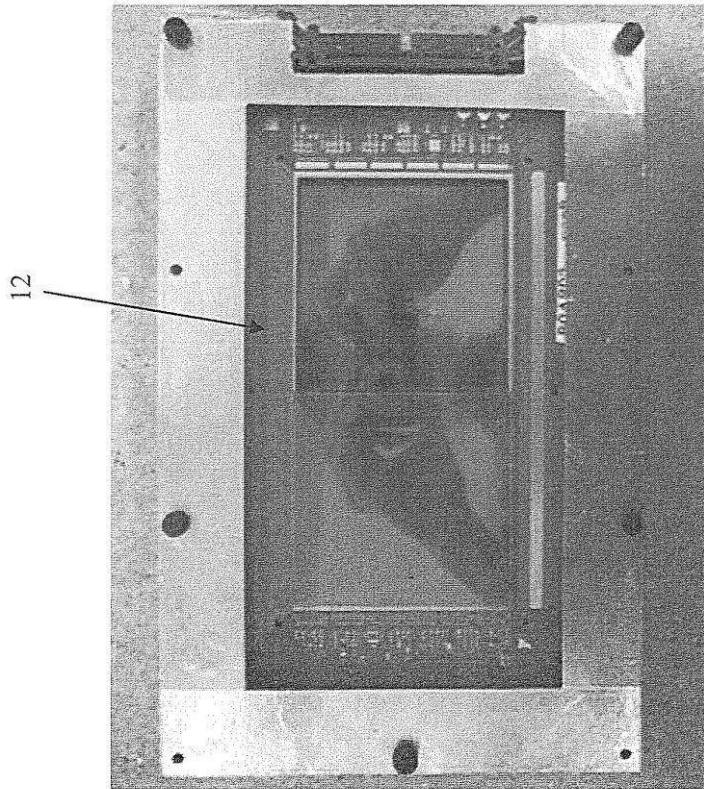


FIGURE 5

【図 6】



B



A

FIGURE 6

【国際調査報告】

INTERNATIONAL SEARCH REPORT

International application No.

PCT/US 11/31104

A. CLASSIFICATION OF SUBJECT MATTER IPC(8) - G01T 1/164 (2011.01) USPC - 250/363.03 According to International Patent Classification (IPC) or to both national classification and IPC		
B. FIELDS SEARCHED Minimum documentation searched (classification system followed by classification symbols) USPC: 250/363.03 Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched IPC(8): G01T 1/164 (2011.01); USPC: 250/361R,362,363.01,363.03,363.04,367,309,492.1 (keyword limited; terms below) Electronic data base consulted during the international search (name of data base and, where practicable, search terms used) PubWEST(USPT,PGPB,EPAB,JPAB); Google Scholar Search Terms: imaging field, detector, sensor, plane, scintillator, fiber, computer, calorimeter, proton, positron, tomography, pct, simpms, graphics, memory, particle, electron, solution, vector, message, passing, 3d, cluster, mlp, mpi, led, stage, rotating, foam, pmma.		
C. DOCUMENTS CONSIDERED TO BE RELEVANT		
Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X — Y	US 2007/0045547 A1 (TUMER) 01 March 2007 (01.03.2007), entire document, especially; FIG 2, 27, 30-32, para [0003], [0006], [0007], [0012], [0013], [0058], [0065], [0066], [0068], [0079], [0080], [0085], [0086], [0091], [0094], [0097], [0099], [0100], [0107], [0109], [0121], [0125], [0148], [0162], [0164], [0166], [0167].	18-20 1-17, 21-26
Y	US 6,635,877 B2 (KUSUYAMA et al.) 21 October 2003 (21.10.2003), entire document, especially; col 2, in 27-61, col 4, in 7-23, col 5, in 1-23.	1-17, 21-26
Y	US 6,078,052 A (DIFILIPPO) 20 June 2000 (20.06.2000), entire document, especially; col 1, in 36-50, col 3, in 30-50, col 5, in 5-34, col 6, in 5-46.	12-14, 22
Y	US 2007/0040830 A1 (PAPAGEORGIOU) 22 February 2007 (22.02.2007), entire document, especially; para [0009], [0028], [0032], [0033], [0035], [0037], [0082].	23-26
A	US 2009/0134334 A1 (NELSON) 28 May 2009 (28.05.2009), entire document.	1-26
☐ Further documents are listed in the continuation of Box C. ☐		
* Special categories of cited documents: "A" document defining the general state of the art which is not considered to be of particular relevance "E" earlier application or patent but published on or after the international filing date "L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified) "O" document referring to an oral disclosure, use, exhibition or other means "P" document published prior to the international filing date but later than the priority date claimed "T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention "X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone "Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art "&" document member of the same patent family		
Date of the actual completion of the international search 30 November 2011 (30.11.2011)		Date of mailing of the international search report 13 DEC 2011
Name and mailing address of the ISA/US Mail Stop PCT, Attn: ISA/US, Commissioner for Patents P.O. Box 1450, Alexandria, Virginia 22313-1450 Facsimile No. 571-273-3201		Authorized officer: Lee W. Young PCT Helpdesk: 571-272-4300 PCT OSP: 571-272-7774

フロントページの続き

(51)Int.Cl.

F I

テーマコード (参考)

G 0 1 T 1/203

(81)指定国 AP(BW,GH,GM,KE,LR,LS,MW,MZ,NA,SD,SL,SZ,TZ,UG,ZM,ZW),EA(AM,AZ,BY,KG,KZ,MD,RU,TJ,TM),EP(AL,AT,BE,BG,CH,CY,CZ,DE,DK,EE,ES,FI,FR,GB,GR,HR,HU,IE,IS,IT,LT,LU,LV,MC,MK,MT,NL,NO,PL,PT,RO,RS,SE,SI,SK,SM,TR),OA(BF,BJ,CF,CG,CI,CM,GA,GN,GQ,GW,ML,MR,NE,SN,TD,TG),AE,AG,AL,AM,AO,AT,AU,AZ,BA,BB,BG,BH,BR,BW,BY,BZ,CA,CH,CL,CN,CO,CR,CU,CZ,DE,DK,DM,DO,DZ,EC,EE,EG,ES,FI,GB,GD,GE,GH,GM,GT,HN,HR,HU,ID,IL,IN,IS,JP,KE,KG,KM,KN,KP,KR,KZ,LA,LC,LK,LR,LS,LT,LU,LY,MA,MD,ME,MG,MK,MN,MW,MX,MY,MZ,NA,NG,NI,NO,NZ,OM,PE,PG,PH,PL,PT,RO,RS,RU,SC,SD,SE,SG,SK,SL,SM,ST,SV,SY,TH,TJ,TM,TN,TR,TT,TZ,UA,UG,US,UZ,VC,VN,ZA,ZM,ZW

(74)代理人 100078282

弁理士 山本 秀策

(74)代理人 100113413

弁理士 森下 夏樹

(72)発明者 キャロニス, ニコラス

アメリカ合衆国 イリノイ 60185, ウェスト シカゴ, チェリー ブロッサム コート 810

(72)発明者 コートラコン, ジョージ

アメリカ合衆国 カリフォルニア 92373, レッドランズ, シャパラル ロード 1748

(72)発明者 ダフィン, カーク

アメリカ合衆国 イリノイ 60115, デカルブ, バープ ブールバード 119

(72)発明者 エルダーリー, ベラ

アメリカ合衆国 イリノイ 60446, ロメオビル, メイフィールド コート 1935

(72)発明者 ナグリック, ケビン

アメリカ合衆国 イリノイ 60123, エルギン, フォックスウィック コート 40 ダブリュー566

(72)発明者 ペンフォード, スコット

オーストラリア国 5000 サウス オーストラリア, アデレード, ロイヤル アデレード ホスピタル, デパートメント オブ メディカル フィジックス, レベル 2 イーストウィング

(72)発明者 ルビノフ, ポール

アメリカ合衆国 イリノイ 60657, バタビア, エムエス 222, フェルミラボ, ピー.オー. ボックス 500

(72)発明者 ライカリン, ビクター

アメリカ合衆国 イリノイ 60502, オーロラ, ウィダウン レーン 2529

(72)発明者 ズチ, ビジュヌ

アメリカ合衆国 イリノイ 60115, デカルブ, ノーザン イリノイ ユニバーシティー, デパートメント オブ フィジックス

Fターム(参考) 2G088 AA01 EE02 FF07 GG10 GG11 GG15 GG18 GG19 JJ05 KK29

KK33