HP OpenVMS Alpha パーティショニングおよび Galaxy ガイド

概要

本書は，OpenVMS Alpha Version 7.3-1 用の『パーティショニングおよび Galaxy ガイド』を置き換えるものです。

HP 部品番号: AA-RFJFE-TE
2009 年 8 月
第 1.1 版

著作権情報

原典

『HP OpenVMS Alpha Partitioning and Galaxy Guide』© 2003 Hewlett-Packard Development Company, L.P.

目次

まえがき....13
本書の対象読者....13
本書の構成....13
関連資料....14
本書で使用する表記法....14

1 パーティションによる負荷の管理....17
1.1 OpenVMS システムでのハード・パーティションとソフト・パーティションの使用....17
1.2 OpenVMS パーティショニングのガイドライン....17
1.2.1 パーティションのセットアップ手順....19
1.3 AlphaServer ES47/ES80/GS1280 でのパーティショニング....22
1.3.1 ハード・パーティションの構成例....22
1.4 AlphaServer GS80/160/320 でのパーティショニング....26
1.4.1 ハード・パーティションの構成例 1....26
1.4.2 ハード・パーティションの構成例 2....28
1.4.3 ハード・パーティションの構成例 3....28
1.4.4 AlphaServer GS80/160/320 システムでのコンソール・ファームウェアの更新....29
1.5 OpenVMS Galaxy サポート....29
1.6 RAD (リソース・アフィニティ・ドメイン) に対する OpenVMS アプリケーション・サポート....31

2 OpenVMS Galaxy の概念....33
2.1 OpenVMS Galaxy の概念およびコンポーネント....33
2.2 OpenVMS Galaxy の機能....35
2.3 OpenVMS Galaxy の利点....36
2.4 OpenVMS Galaxy バージョン 7.3 の機能....37
2.5 OpenVMS Galaxy の利点....38
2.6 OpenVMS Galaxy が最良の選択ではない場合....38
2.7 さまざまな OpenVMS Galaxy 構成....38
2.7.1 shared-nothing コンピューティング・モデル....39
2.7.2 shared-partial コンピューティング・モデル....39
2.7.3 shared-everything コンピューティング・モデル....40
2.8 シングル・インスタンス Galaxy 構成....40
2.9 OpenVMS Galaxy の構成上の考慮点....40
2.9.1 XMI バスのサポート....41
2.9.2 メモリ分割に関する制限事項....41
2.9.3 EISA バスのサポート....41
2.10 推奨される CD ドライブ....41
2.11 Galaxy を使ったクラスタ....41
2.11.1 OpenVMS Galaxy インスタンスになる....42
2.11.2 SCSI Cluster に関する留意事項....42
2.12 OpenVMS Galaxy コンピューティング環境でのセキュリティに関する留意事項....43
2.13 OpenVMS Galaxy インスタンスのタイム・ゾーンでの構成....43
2.14 OpenVMS Galaxy プログラムの開発....43
2.14.1 ロック・プログラミング・インタフェース....43
2.14.2 システム・イベント・プログラミング・インタフェース....44
2.14.3 OpenVMS Galaxy での System Dump Analyzer (SDA) の使用....44
2.14.3.1 共用メモリのダンプ....45
2.14.3.2 SDA コマンド・インタフェースの変更または追加についてのまとめ....45

3 OpenVMS アプリケーションに対する NUMA の影響.....47
3.1 OpenVMS の NUMA への対応.....47
3.1.1 ホーム RAD.....48
3.1.2 システム・コードの複製.....48
3.1.3 グローバル・ページの分散.....48
3.2 アプリケーション・リソースに関する注意事項.....48
3.2.1 プロセスと共用データ.....49
3.2.2 メモリ.....49
3.2.3 共用と同期処理.....49
3.2.4 OpenVMS 機能の使用.....49
3.3 NUMA リソース・アフィニティ・ドメインのバッチ・ジョブ・サポート.....49
3.3.1 バッチ・キュー・レベルの RAD サポート.....50
3.3.1.1 例.....50
3.3.2 ジョブ・レベルの RAD サポート.....51
3.3.2.1 例.....51
3.3.3 実行時の動作.....51
3.3.3.1 エラー処理.....51
3.3.3.2 バッチ・キューの RAD の変更.....52
3.4 RAD アプリケーション・プログラミング・インタフェース.....52
3.5 RAD システム・サービスの要約の表.....53
3.6 RAD DCL コマンドの要約の表.....53
3.7 System Dump Analyzer (SDA) の RAD サポート.....53
3.7.1 SHOW RAD.....53
3.7.2 SHOW RMD (予約済みメモリ記述子).....55
3.7.3 SHOW PFN.....55
3.7.4 RAD のハード・アフィニティのサポート.....55

4 AlphaServer GS140/GS60/GS60E システムでの OpenVMS Galaxy の構築.....57
4.1 ファームウェアのダウンロード.....57
4.2 OpenVMS Galaxy の構築.....57

5 AlphaServer 8400 システムでの OpenVMS Galaxy の構築.....59
5.1 ステップ 1: 構成の選択とハードウェア要件の判断.....59
5.2 ステップ 2: ハードウェアの設定.....59
5.2.1 KFE72-DA コンソール・サブシステム・ハードウェアの概要.....59
5.2.2 KFE72-DA モジュールの取り付け.....60
5.2.2.1 PCI カードをケージから取り出す.....60
5.2.2.2 モジュールを挿入し，リボン・ケーブルを接続する.....60
5.2.2.3 コネクタの接続.....61
5.2.3 シェルフをシステムに取り付ける.....62
5.2.4 ターミナル・サーバの使用.....62
5.2.5 EISA 装置の取り付け.....62
5.3 ステップ 3: システム・ディスクの作成.....63
5.4 ステップ 4: OpenVMS Alpha のインストール.....63
5.4.1 OpenVMS Galaxy ライセンス情報.....63
5.5 ステップ 5: ファームウェアのアップグレード.....63
5.6 ステップ 6: 環境変数の設定.....64
5.7 ステップ 7: セカンダリ・コンソール装置の起動.....65
5.8 ステップ 8: OpenVMS Galaxy のブート.....67

6 AlphaServer 8200 システムでの OpenVMS Galaxy の構築.....69
6.1 ステップ 1: 構成の選択とハードウェア要件の判断.....69
6.2 ステップ 2: Galaxy ハードウェアの設定.....69
6.2.1 EISA 装置の取り付け.....69
6.3 ステップ 3: システム・ディスクの作成.....70
6.4 ステップ 4: OpenVMS Alpha バージョン 7.3 のインストール.....70
6.5 ステップ 5: ファームウェアのアップグレード.....70
6.6 ステップ 6: 環境変数の設定.....71
6.7 ステップ 7: セカンダリ・コンソール装置の起動.....72
6.8 ステップ 8: OpenVMS Galaxy のブート.....73

7 AlphaServer 4100 システムでの OpenVMS Galaxy の構築.....75
7.1 はじめに.....75
7.2 ステップ 1: AlphaServer 4100 構成を確かめる.....76
7.3 ステップ 2: OpenVMS Alpha Version 7.3-2 をインストールする.....76
7.4 ステップ 3: ファームウェアをアップグレードする.....77
7.5 ステップ 4: 環境変数を設定する.....77
7.6 ステップ 5: システムを初期化し，コンソール装置を起動する.....77

8 AlphaServer ES40 システムでの OpenVMS Galaxy の構築.....81
8.1 コンピューティング環境を構築する前に.....81
8.2 ステップ 1: AlphaServer ES40 の構成を確かめる.....82
8.3 ステップ 2: OpenVMS Alpha バージョン 7.3-2 をインストールする.....83
8.4 ステップ 3: ファームウェアをアップグレードする.....84
8.5 ステップ 4: 環境変数の設定.....84
8.6 ステップ 5: システムを初期化し，コンソール装置を起動する.....85

9 AlphaServer GS80/160/320 システムでの OpenVMS Galaxy の構築.....87
9.1 ステップ 1: 構成の選択とハードウェア要件の判断.....87
9.2 ステップ 2: ハードウェアの設定.....87
9.3 ステップ 3: システム・ディスクの作成.....87
9.4 ステップ 4: OpenVMS Alpha バージョン 7.3-2 のインストール.....87
9.4.1 OpenVMS Galaxy ライセンス情報.....87
9.5 ステップ 5: 環境変数の設定.....88
9.5.1 AlphaServer GS160 の例.....88
9.5.2 AlphaServer GS320 の例.....88
9.5.3 環境変数の説明.....89
9.6 ステップ 6: セカンダリ・コンソール装置の起動.....91
9.7 ステップ 7: セカンダリ・コンソールの初期化.....91
9.8 ステップ 8: OpenVMS Galaxy のブート.....92

10 AlphaServer ES47/ES80/GS1280 システムでの OpenVMS Galaxy の構築.....93
10.1 ステップ 1: 構成の選択とハードウェア要件の判断.....93
10.2 ステップ 2: ハードウェアの設定.....93
10.3 ステップ 3: システム・ディスクの作成.....93
10.4 ステップ 4: OpenVMS Alpha のインストール.....93
10.4.1 OpenVMS Galaxy ライセンス情報.....94
10.5 Step 5: パーティションの設定.....94
10.6 ステップ 6: OpenVMS Galaxy のブート.....102

11 Alpha システムでのシングル・インスタンス Galaxy の使用.....105
11.1 GCU を使ったシングル・インスタンス Galaxy の作成.....105
11.2 Galaxy インスタンスとしてのリブート.....105

12 OpenVMS Galaxy に関するヒントと手法.....107
12.1 システム・オート・アクション.....107
12.2 コンソール環境変数の変更.....107
12.3 コンソールに関するヒント.....107
12.4 Galaxy モードのオフ設定.....108

13 OpenVMS Galaxy Configuration ユーティリティ.....109
13.1 GCU の概略.....110
13.1.1 Galaxy 構成モデルの作成.....110
13.1.2 監視.....111
13.1.2.1 レイアウト管理.....111
13.1.2.2 OpenVMS Galaxy チャート.....112
13.1.3 会話.....112
13.2 GCU による OpenVMS Galaxy の管理.....113
13.2.1 独立インスタンス.....113
13.2.2 分離されたインスタンス.....113
13.2.3 必要な PROXY アクセス.....113
13.3 Galaxy 構成モデル.....114
13.3.1 アクティブ・モデル.....115
13.3.2 オフライン・モデル.....115
13.3.2.1 例: オフライン・モデルの作成.....115
13.4 GCU チャートの使用.....116
13.4.1 コンポーネントの識別と表示プロパティ.....116
13.4.2 Physical Structure チャート.....117
13.4.2.1 ハードウェア・ルート.....117
13.4.2.2 所有権オーバーレイ.....117
13.4.3 Logical Structure チャート.....117
13.4.3.1 ソフトウェア・ルート.....118
13.4.3.2 未割り当てリソース.....118
13.4.3.3 コミュニティ・リソース.....118
13.4.3.4 インスタンス・リソース.....118
13.4.4 Memory Assignment チャート.....118
13.4.4.1 コンソール・フラグメント.....119
13.4.4.2 プライベート・フラグメント.....119
13.4.4.3 共用メモリ・フラグメント.....119
13.4.5 CPU Assignment チャート.....119
13.4.5.1 プライマリ CPU.....119
13.4.5.2 セカンダリ CPU.....119
13.4.5.3 Fast Path およびアフィニティされた CPU.....119
13.4.5.4 失われた CPU.....119
13.4.6 IOP Assignment チャート.....120
13.4.7 Failover Target チャート.....120
13.5 コンポーネント・パラメータの表示.....121
13.6 コンポーネント・コマンドの実行.....121
13.7 GCU メニューのカスタマイズ.....121
13.8 HP Availability Manager による OpenVMS Galaxy の監視.....122
13.9 CPU Load Balancer プログラムの実行.....122
13.10 インスタンスの作成.....123
13.11 インスタンスの破棄.....123

13.12 シャットダウンおよびリブート・サイクル....123
13.13 オンライン・モデルとオフライン・モデル....123
13.14 GCU システム・メッセージ....123

14 Graphical Configuration Manager....129
14.1 概要....129
14.2 インストールの前提条件....130
14.2.1 ソフトウェア要件....130
14.2.2 ハードウェア要件....130
14.3 インストール手順....130
14.3.1 キット....131
14.3.2 GCM サーバのインストール....131
14.3.3 GCM クライアントのインストール....132
14.3.3.1 OpenVMS GCM クライアント....133
14.3.3.2 PC GCM クライアント....134
14.3.4 GCM サーバのセットアップ....134
14.4 GCM サーバの起動....138
14.4.1 GCM サーバの自動起動....138
14.4.2 GCM サーバの手動起動....139
14.5 インストール後の管理タスク....139
14.5.1 GCM ユーザの登録とサブスクリプション・テンプレートの定義....139
14.5.2 ライブラリの管理 140
14.5.3 カスタム・バナー....141
14.5.4 システム通知....141
14.5.5 コマンドのカスタマイズ....141
14.6 アソシエーションの構成....141
14.6.1 考慮事項....142
14.6.2 アソシエーションへのシステムの追加....142
14.7 GCM のカスタマイズ....143
14.7.1 コマンドの定義と使用....143
14.8 GCM サーバのログ・ファイル....144
14.9 GCM サーバのトラブルシューティング....145
14.9.1 診断情報の取得....145
14.9.2 潜在的な問題領域....146
14.9.3 タイムアウトの検出....146
14.10 性能....146
14.11 GCM サーバの保守....147
14.12 冗長な GCM サーバ・セットアップの例....147

15 DCL CLI を使った CPU の再割り当て....153
15.1 DCL による再割り当て....153
15.2 GCU のドラッグ・アンド・ドロップによる再割り当て....153
15.3 インターモダル再割り当て....153
15.4 Galaxy サービスを使用したソフトウェアによる再割り当て....154
15.5 再割り当ての失敗....154

16 Galaxy と NUMA のコマンドとレキシカル関数....157
16.1 CPU 関連の DCL コマンド....157
16.2 Galaxy 関連の DCL レキシカル関数....158
16.3 DCL コマンド例....161
16.3.1 CPU コマンド....161
16.3.1.1 STOP/CPU/MIGRATE....161
16.3.1.2 SHOW CPU....161

16.3.1.3 SET CPU....162
16.3.2 SHOW MEMORY....163
16.3.3 レキシカル関数の例....163
16.3.4 INSTALL LIST....163
16.3.5 INSTALL ADD/WRITABLE....164
16.3.6 CONFIGURE GALAXY....164

17 共用メモリを使った通信....167
17.1 共用メモリ・クラスタ・インターコネクト (SMCI)....167
17.1.1 SYS$PBDRIVER ポート・デバイス....167
17.1.2 1 つの Galaxy 内の複数のクラスタ....167
17.1.3 SYS$PBDRIVER の SYSGEN パラメータ....168
17.1.3.1 SMCI_PORTS....168
17.1.4 SMCI_FLAGS....168
17.2 LAN 共用メモリ・デバイス・ドライバ....169

18 共用メモリ・プログラミング・インタフェース....171
18.1 共用メモリの使用....171
18.2 システム・サービス....171
18.2.1 強化されたサービス....172
18.2.2 新しいセクション・フラグ SEC$M_READ_ONLY_SHPT....172
18.3 Galaxy-wide グローバル・セクション....173

19 OpenVMS Galaxy デバイス・ドライバ....175
19.1 ダイレクト・マップ DMA ウィンドウの変更....175
19.2 OpenVMS バージョン 7.2 より以前に PCI ダイレクト・マップ DMA がどのように機能していたか....175
19.3 現在のバージョンの OpenVMS で PCI ダイレクト・マップ DMA がどのように機能するか....175
19.4 0 以外のダイレクト・マップ DMA ウィンドウをサポートするための IOC$NODE_DATA の変更....176

A OpenVMS Galaxy CPU Load Balancer プログラム....177
A.1 CPU Load Balancer の概要....177
A.1.1 必要な特権....177
A.1.2 構築とコピーの手順....177
A.1.3 スタートアップ・オプション....177
A.2 プログラム例....178

B メモリ・サイズ設定の共通値....189

C ライセンスのインストール....191
C.1 ライセンス・プロセス....191
C.1.1 ライセンスについての詳細情報....192

用語集....193

索引....195

図目次

1-1 パーティショニングの手順....20
1-2 システム構成ツリー....20
1-3 ハードウェア構成ツリー....21
1-4 ソフトウェア構成のツリー....22
1-5 構成例 1....26
1-6 構成例 2....28
1-7 構成例 3....29
1-8 メモリの使われ方....30
1-9 構成ツリーで表したソフト・パーティショニング....31
2-1 OpenVMS Galaxy アーキテクチャの概念図....34
2-2 別の Galaxy アーキテクチャ・ダイアグラム....35
2-3 shared-nothing コンピューティング・モデル....39
2-4 shared-partial コンピューティング・モデル....39
2-5 shared-everything コンピューティング・モデル....40
5-1 リボン・ケーブルの接続....61
5-2 コネクタ....62
19-1 PCI ベース DMA....175
19-2 OpenVMS DMA....176

表目次

1-1 システム・ビルディング・ブロック....18
2-1 ロック・プログラミングのための Galaxy システム・サービス....44
2-2 イベント・プログラミングのための Galaxy システム・サービス....44
2-3 DUMPSTYLE のビットの定義....45
2-4 SDA コマンドの拡張....46
10-1 ES47/ES80/GS1280 の構成....93
14-1 単純なアソシエーションのワークシート例....142
16-1 CPU 関連の DCL コマンド....157
16-2 NUMA/RAD 関連の DCL コマンド....158
16-3 Galaxy 関連の DCL コマンド....158
16-4 Galaxy F$GETSYI 項目コード....158
16-5 パーティショニング F$GETSYI 項目コード....159
16-6 SMP F$GETSYI 項目コード....159
16-7 RAD F$GETJPI 項目コード....160
16-8 RAD F$GETSYI 項目コード....160
B-1 メモリ・サイズ環境変数の共通値....189

例目次

14-1 GCM サーバのインストール例.....132
14-2 GCM サーバ・ファイルのディレクトリ.....132
14-3 OpenVMS GCM クライアントのインストール.....133
14-4 GCM クライアント・ファイルのディレクトリ.....134
14-5 GCM サーバのセットアップ例.....136

まえがき

『パーティショニングおよび Galaxy ガイド』は，OpenVMS Alpha バージョン 7.3-2 で提供されるパーティショニングと OpenVMS Galaxy 機能を活用する方法について説明します。

本書に説明されている情報は OpenVMS Alpha システムにのみ適用され，OpenVMS VAX システムには適用されません。

本書の対象読者

本書は，システム管理者，アプリケーション・プログラマ，技術コンサルタント，データ・センター管理者をはじめ，OpenVMS Alpha の OpenVMS Galaxy および パーティショニング機能に関する知識を必要とする方を対象にしています。

本書の構成

『パーティショニングおよび Galaxy ガイド』は，OpenVMS パーティショニングをサポートするハードウェア・プラットフォームでの OpenVMS パーティショニングの概念と機能を紹介します。また，OpenVMS Alpha バージョン 7.3-2 で提供される OpenVMS Galaxy 機能の使用方法についても説明します。

本書は，次の章と付録で構成されています。

第1章「パーティションによる負荷の管理」では，ハード・パーティションおよびソフト・パーティションの使用方法と，RAD (リソース・アフィニティ・ドメイン) に対する OpenVMS のサポートについて説明します。

第2章「OpenVMS Galaxy の概念」では，OpenVMS Galaxy の概念を説明し，OpenVMS バージョン 7.3-2 で使用可能な機能の概要を述べます。

第3章「OpenVMS アプリケーションに対する NUMA の影響」では，OpenVMS アプリケーションに対する NUMA (nonuniform memory access) の影響について述べます。

第4章「AlphaServer GS140/GS60/GS60E システムでの OpenVMS Galaxy の構築」では，AlphaServer GS140/GS60/GS60E システムでの OpenVMS Galaxy の作成方法を説明します。

第5章「AlphaServer 8400 システムでの OpenVMS Galaxy の構築」では，AlphaServer 8400 システムでの OpenVMS Galaxy の作成方法を説明します。

第6章「AlphaServer 8200 システムでの OpenVMS Galaxy の構築」では，AlphaServer 8200 システムでの OpenVMS Galaxy の作成方法を説明します。

第7章「AlphaServer 4100 システムでの OpenVMS Galaxy の構築」では，AlphaServer 4100 システムでの OpenVMS Galaxy の作成方法を説明します。

第8章「AlphaServer ES40 システムでの OpenVMS Galaxy の構築」では，AlphaServer ES40 システムでの OpenVMS Galaxy の作成方法を説明します。

第9章「AlphaServer GS80/160/320 システムでの OpenVMS Galaxy の構築」では，AlphaServer GS80/160/320 システムでの OpenVMS Galaxy の作成方法を説明します。

第10章「AlphaServer ES47/ES80/GS1280 システムでの OpenVMS Galaxy の構築」では，AlphaServer ES47/ES80/GS1280 システムでの OpenVMS Galaxy の作成方法を説明します。

第11章「Alpha システムでのシングル・インスタンス Galaxy の使用」では，Alpha システムでのシングル・インスタンス Galaxy の使用方法を説明します。

第12章「OpenVMS Galaxy に関するヒントと手法」では，OpenVMS Galaxy に関するヒントと手法について説明します。

第13章「OpenVMS Galaxy Configuration ユーティリティ」では，OpenVMS Galaxy Configuration ユーティリティについて説明します。

第14章「Graphical Configuration Manager」では，OpenVMS Graphical Configuration Manager について説明します。

第15章「DCL CLI を使った CPU の再割り当て」では， CPU の再割り当てについて説明します。

第16章「Galaxy と NUMA のコマンドとレキシカル関数」では， OpenVMS Galaxy の管理に役立つ DCL コマンドについて説明します。

第17章「共用メモリを使った通信」では， 共用メモリとの通信について説明します。

第18章「共用メモリ・プログラミング・インタフェース」では， 共用メモリ・プログラミング・インタフェースについて説明します。

第19章「OpenVMS Galaxy デバイス・ドライバ」では， OpenVMS Galaxy デバイス・ドライバについて説明します。

付録 A「OpenVMS Galaxy CPU Load Balancer プログラム」には， OpenVMS Galaxy 負荷分散プログラムの例があります。

付録 B「メモリ・サイズ設定の共通値」には，メモリ・サイズの設定に関する共通値の一覧があります。

付録 C「ライセンスのインストール」では，ライセンス管理について説明します。

本書では， 読者が OpenVMS の概念と操作について十分理解しているものと想定しているため， OpenVMS に関する基本情報は記載していません。

関連資料

次のマニュアルには，パーティション分割されたコンピューティング環境に役立つ OpenVMS の情報が含まれています。

- 『OpenVMS インストレーション・ガイド [翻訳版]』
- 『OpenVMS Cluster システム』
- 『OpenVMS Alpha System Analysis Tools Manual』
- 『OpenVMS License Management Utility Manual』

HP OpenVMS 製品とサービスについての詳細は， 次の Web サイトを参照してください。

http://www.hp.com/go/openvms/

または

http://www.hp.com/jp/openvms/

本書で使用する表記法

本書では次の表記法を使用しています。

表記法	意味
Ctrl/**x**	Ctrl/**x** という表記は，Ctrl キーを押しながら別のキーまたはポインティング・デバイス・ボタンを押すことを示します。
PF1 **x**	PF1 **x** という表記は，PF1 に定義されたキーを押してから， 別のキーまたはポインティング・デバイス・ボタンを押すことを示します。
Return	例の中で，太字のキー名は，そのキーを押すことを示します。
...	例の中の水平方向の反復記号は，次のいずれかを示します。 • 文中のオプションの引数が省略されている。 • 前出の 1 つまたは複数の項目を繰り返すことができる。 • パラメータや値などの情報をさらに入力できる。
. . .	垂直方向の反復記号は，コードの例やコマンド形式の中の項目が 省略されていることを示します。このように項目が省略されるのは， 説明している内容にとってその項目が重要ではないからです。
( )	コマンドの形式の説明において，括弧は， 複数の項目を指定する場合に， 選択した項目を括弧で囲まなければならないことを示しています。

表記法	意味
[ ]	コマンドの形式の説明において，大括弧で囲まれた要素は省略可能であることを示します。 項目を 1 つ以上選択しても， あるいは 1 つも選択しなくても構いません。コマンド行では大括弧は入力しないでください。ただし，OpenVMS のディレクトリ指定の構文や，割り当て文の部分文字列指定の構文の中の大括弧は，含めなければなりません。
|	コマンド形式の説明では，縦線は大括弧や中括弧内の選択項目を区切ります。大括弧内の選択肢は省略可能ですが，中括弧内の選択肢は少なくとも 1 つ選択する必要があります。 コマンド行では縦線は入力しないでください。
{ }	コマンドの形式の説明において，中括弧で囲まれた項目は必須の選択肢です。いずれか 1 つの項目を選択しなければなりません。 コマンド行では中括弧は入力しないでください。
太字体	太字体のテキストは，新しい用語を示します。また，引数，属性，条件の名前を示すときにも使用されます。
`italic type`	イタリック体のテキストは，重要な情報を示します。 また，システム・メッセージ (たとえば内部エラー `number`)， コマンド行 (たとえば /PRODUCER=`name`)， コマンド・パラメータ (たとえば `device-name`) などの変数を示す場合にも使用されます。
UPPERCASE TYPE	英大文字のテキストは，コマンド，ルーチン名，ファイル名， ファイル保護コード名，システム特権の短縮形を示します。
`Example`	この書体は，コード例，コマンド例および会話型の画面表示を示します。 また，テキスト中のこの書体は，URL，UNIX のコマンドとパス名，PC ベースのコマンドとフォルダ，C プログラミング言語の言語要素などを示します。
-	コマンド形式の記述の最後，コマンド・ライン，コード・ラインにおいて， ハイフンは，要求に対する引数がその後の行に続くことを示します。
数字	特に明記しない限り，本文中の数字はすべて 10 進数です。 10 進数以外 (2 進数，8 進数，16 進数) は，その旨を明記してあります。

第1章 パーティションによる負荷の管理

OpenVMS のユーザは，ハードおよびソフト・パーティションをサポートするシステムをさまざまな形で使用しています。これらのシステムを最も効果的に使用するためには，コンピューティングとアプリケーションに対するユーザのニーズに最適な構成オプションを検討する必要があります。

この章では，ハード・パーティションとソフト・パーティションの使用方法，および，新しい AlphaServer システムでできるだけ効率良くアプリケーションを実行させる **RAD (リソース・アフィニティ・ドメイン)** に対する OpenVMS サポートについて説明します。

1.1 OpenVMS システムでのハード・パーティションとソフト・パーティションの使用

ハード・パーティショニングとは，ハードウェアで強制されたアクセス・バリアで コンピューティング・リソースを物理的に分離したものです。ハード・パーティション境界を越えて読み込みや書き込みを行うことはできず，ハード・パーティション同士ではリソースは共用されません。

ソフト・パーティショニングとは，ソフトウェアで制御されたアクセス・バリアでコンピューティング・リソースを分離したものです。 ソフト・パーティション (サブパーティションと呼ぶこともあります) では，複数のオペレーティング・システム間でハードウェア・リソースを共用できます。 ソフト・パーティション境界を越えた読み込みと書き込みのアクセスは， オペレーティング・システムまたはアプリケーションによって制御されます。 OpenVMS Galaxy は，ソフト・パーティショニングの実装です。

新しい AlphaServer ES シリーズまたは GS シリーズのシステムでどのようなパーティション分割を選択するかは，コンピューティング環境とアプリケーションの要件によって決まります。パーティショニングを計画するときには，アプリケーションに必要なメモリ量と，どのオペレーティング・システムを動作させるかを考慮する必要があります。パーティショニングをサポートする OpenVMS システムの構成方法を決定するときには， 次の項目を検討します。

- 必要なハード・パーティションの数
- 必要なソフト・パーティションの数
- パーティションのサイズをどの程度小さくできるか

  複数のハード・パーティションを使用することで，パーティション間でのハードウェア・セキュリティが最も高くなります。ハード・パーティション上で 1 つのソフト・パーティションを動作させたときには，専用のマシンで動作しているのと同等になります。

  1 つのハード・パーティション上で複数のソフト・パーティションを動作させると，CPU やメモリなどのリソースが共用でき，性能面でメリットがあります。

1.2 OpenVMS パーティショニングのガイドライン

ハード・パーティショニングは，AlphaServer ES47/ES80/GS1280 および GS80/160/320 システムでのみ利用できます。 AlphaServer ES47/ES80/GS1280 システム上でのパーティションの利用は，GS80/160/320 システムでパーティションを利用するのと同様です。

AlphaServer ES または GS シリーズ・システムでハード・パーティションやソフト・パーティションを使用するかどうかを決定するときには，次の点に注意してください。

- AlphaServer GS80/160/320 では，各パーティション (ハードまたはソフト) は，それぞれコンソール・ラインを持っている必要があります。 AlphaServer ES47/ES80 および GS1280 システムでは，コンソールへのネットワーク接続または直接アクセスが必要です。 AlphaServer GS80/160/320 では，システムの QBB ごとにコンソール・ラインを 1 つずつ持つことができます。
- 1 つのハード・パーティションには，複数のソフト・パーティションを作成できます。

- ライセンシング・ポリシの例として，AlphaServer ES または GS シリーズのシステム全体でクラスタ・ライセンスは 1 つしか必要ありません。 インスタンスがいくつあり，それが内部的または外部的にどのようにクラスタ接続されるかには関係しません。 ライセンスについては，本書の 付録 C「ライセンスのインストール」 と，ライセンシング・ポリシ自体を参照してください。

注意:

OpenVMS Galaxy を使ったコンピューティング環境では，MOP (Maintenance Operations Protocol) ブートはインスタンス 0 でしかサポートされません。

- ハード・パーティションは，クォド・ビルディング・ブロック (QBB) またはシステム・ビルディング・ブロック (SBB) のビルディング・ブロック境界に作成します。 GS80/160/320 システムでは QBB が使われ，ES47/ES80/GS1280 システムでは SBB が使われます。 1 つのハード・パーティションには複数の SBB または QBB を含めることができます。 可用性を高めるため，ハード・パーティションは SBB 境界に作成することをお勧めします。 1 つのパーティションに 32 個より多くのプロセッサを含めることはできません。

  サブシステム・ビルディング・ブロック (SSBB) は，GS1280 でのみ使用できます。 GS1280 の SSBB は，8P ビルディング・ブロックの中の 2P ビルディング・ブロックです。

  AlphaServer ES47/ES80 システムでは，2 プロセッサ (2P) の SBB が使われ，GS1280 では 8 プロセッサ (8P) の SBB が使われます。 2P SBB には I/O が含まれますが，8P SBB では，外部の I/O 接続を準備する必要があります。 SSBB と SBB はどちらも，ハード・パーティションの障害分離のために使用できます。 2 種類の SBB とそのバリエーションについては，表 1-1 「システム・ビルディング・ブロック」 を参照してください。
- ソフト・パーティションはビルディング・ブロック境界に置く必要はありません。

表 1-1 システム・ビルディング・ブロック

システム	SBB の種類	モデル	ハード・パーティションの最大数
ES47/ES80	2P (2x1)	2	1
	2P (2x2)	4	2
ES80	2P (2x3)	6	3
	2P (2x4)	8	4
GS1280	8P (8x1)	8	4 - 2P SSBB 使用時
	8P (8x2)	16	8 - 2P SSBB 使用時
	8P (8x1)	8	1 - 8P SBB 使用時
	8P (8x2)	16	2 - 8P SBB 使用時
	8P (8x4)	32	4 - 8P SBB 使用時

パーティションのセットアップでは，セットアップ規則に従わなければなりません。

QBB ベースのシステムでは，パーティション (ハードまたはソフト) それぞれにコンソール・ラインを持つ必要があります。 システム内の各 QBB はコンソール・ラインを 1 つ持つことができます。 したがって，システムのパーティション (ハードまたはソフト) の最大数は，システムの QBB の数と同じになります。

ハード・パーティションの規則

各ハード・パーティションは，次のものを備えている必要があります。

- マネージメント・バックプレーン・モジュール (MBM) または telnet によるコンソールへのアクセス
- パーティションごとに 1 つの I/O ドローワ (ES47/ES80 では 内部ドローワでよい)。
- 単一点障害を避けるため，ビルディング・ブロック境界で分割する必要があります。

障害分離性と可用性を最大限に高めるため，ES47/ES80/GS1280 システムでは，システム・ビルディング・ブロック境界でハード・パーティションに分割します。 これは，ES47/ES80 では 2P SBB 境界になり，GS1280 では 8P SBB 境界になります。 システム・ビルディング・ブロック境界でハード・パーティションに分割することで，システム全体で単一点障害がなくなります。 電源と冷却装置も，そのハード・パーティションに含まれています。 プロセッサ間のリンクは遮断されます。 このようにパーティショニングを行うのが最も堅牢です。 1 つの堅牢なハード・パーティション内には複数のシステム・ビルディング・ブロックを含めることができます。

GS1280 システムでは，システム・ビルディング・ブロックをサブシステム・ビルディング・ブロック (SSBB) 単位のハード・パーティションに分割できます。 1 つの 8P SBB を 2P レベルに分割できます。 これらのハード・パーティションは個別に電源が供給され，修理が必要なときにはデュアル CPU モジュールの電源を切ることができます。 このパーティショニングのレベルでは，8P SBB 境界で分割されたシステムが持つ障害分離性や堅牢性はありません。

8P または 2P の SBB レベルでのハード・パーティションでは，ハード・パーティションごとに個別のシリアル・コンソール・ラインがサポートされます。 1 つの 8P SBB を複数のハード・パーティションに分割すると，シリアル・コンソールは同時に 1 つのサブパーティションにしか接続できません。 すべてのサブパーティションのコンソールに同時にアクセスする必要がある場合には，管理用の LAN を使った telnet セッションを使用しなければなりません。 1.2.1 項 「パーティションのセットアップ手順」では，パーティションのセットアップ手順を説明しています。 また，1.3.1 項 「ハード・パーティションの構成例」では，2 つの例を挙げて構成のセットアップ方法を説明しています。

注意:

必要なコンソール

ES47/ES80/GS1280 シリーズのハード・パーティション機能では，最低限 V6.6 のコンソール・セットが必要です。 このコンソール・セットは，次の AlphaServer ファームウェアの Web サイトから入手できます。

http://ftp.digital.com/pub/Digital/Alpha/firmware/readme.html

この Web サイト・アドレスでは大文字と小文字が区別されることに注意してください。

ソフト・パーティションの規則

各ソフト・パーティションには次のものが含まれている必要があります。

- MBM または telnet を使ったコンソールへのアクセス
- パーティションごとに 1 つの I/O ドローワ (ES47/ES80 では内部ドローワでよい)
- 1 基 のプライマリ CPU
- オペレーティング・システムとアプリケーションで使用するプライベート・メモリと共用メモリ。 共用メモリは独立したインスタンスには不要だが，共用メモリを使うアプリケーションには必要

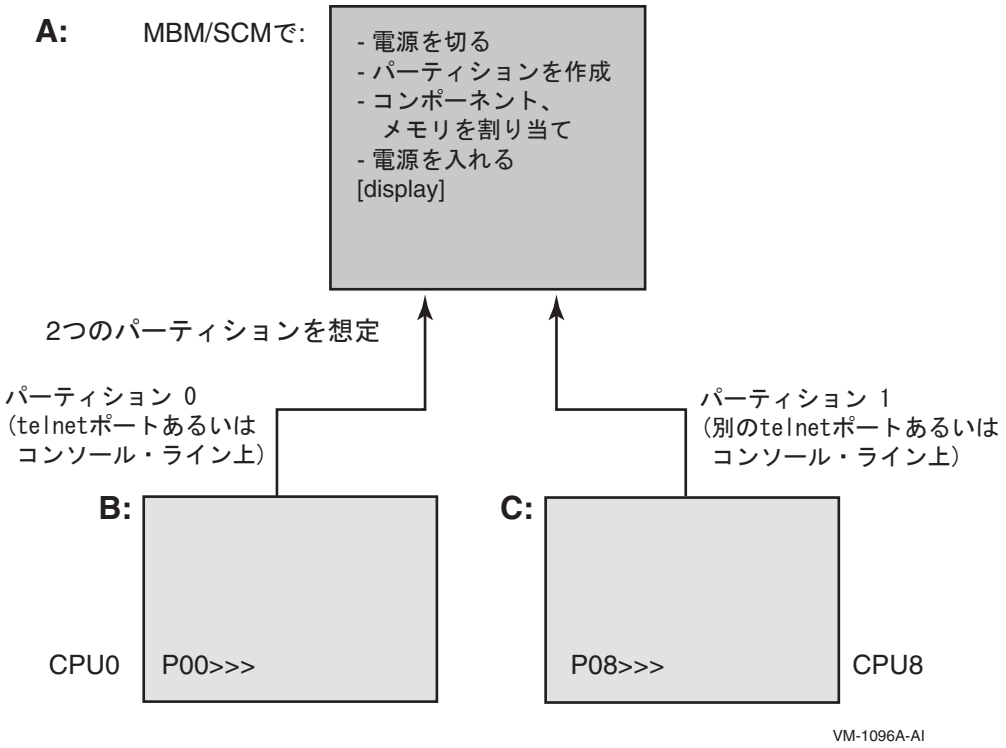
1.2.1 パーティションのセットアップ手順

マネージメント・バックプレーン・モジュール (MBM または SCM) でハード・パーティションおよびソフト・パーティションをセットアップする基本的な手順は，次のとおりです。

1. ハード・パーティションおよびソフト・パーティションを作成する。
2. CPU，IO，およびメモリ・リソースをパーティションに割り当てる。
3. CPU の電源を投入し，コンソールに接続する。
4. コンソールの環境変数を確認し，必要に応じて再設定する。

この手順を図 1-1 「パーティショニングの手順」 に示します。

図 1-1 パーティショニングの手順



物理的なハードウェアと所有関係を，システムごとに単一構成ツリーのブランチとして表します。 パーティション (ハードとソフトの両方) は，その構成内のすべてのリソースに対してアクセスできるかどうかを示す所有関係のコンテナと見なすことができます。 図 1-2「システム構成ツリー」の最上位の構成のブランチには，ハードウェアとソフトウェアの両方の構成ツリーが含まれます。

図 1-2 システム構成ツリー

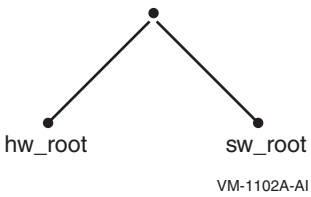
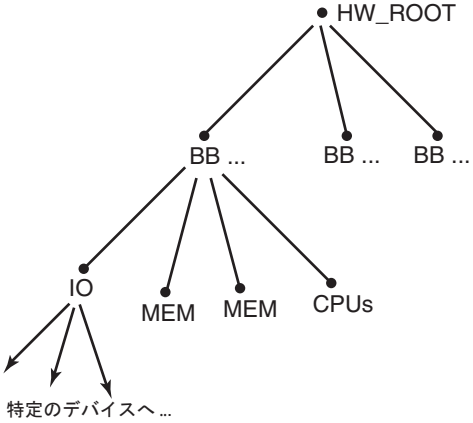


図 1-3「ハードウェア構成ツリー」のハードウェア構成ツリーでは，物理的構成に従って箱の要素が区切られています。黒い丸は，ツリー中のノードを表します (ノードは個別のコンピュータではなく，ツリー中の接続ポイントです)。 ハードウェアのルートから，ツリーはビルディング・ブロックに分かれます。 各ビルディング・ブロックは，メモリ，I/O，CPU といった主要なシステム・カテゴリに分かれます。構成ツリーの下位のレベル，たとえば I/O の中は，システム上の個別のデバイスに分かれます。ツリーの各ノードは，親，兄弟，子，所有関係といった定義を持っています。

図 1-3 ハードウェア構成ツリー



VM-1097A-AI

1 つのパーティションの範囲は必ず 1 つの枝の部分で，その上下部分を含みますが，枝にまたがることはできません。 図 1-4「ソフトウェア構成のツリー」に示すように，ソフト・パーティションは，常にツリーを上にたどって利用できるリソースを探します。ハード・パーティションは，その下のすべてのノードで利用できるように割り当てられたリソースを所有します。

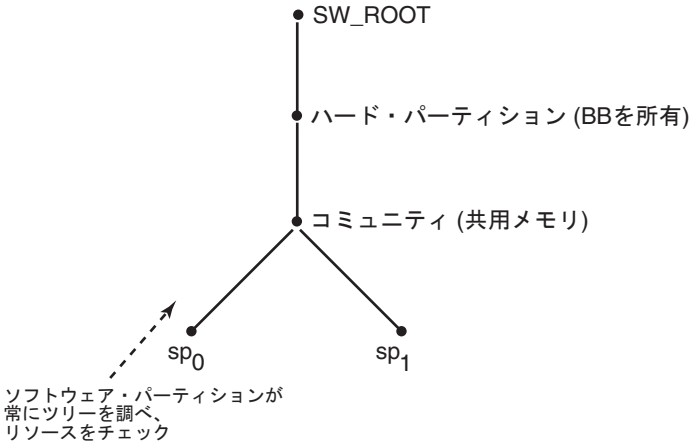
一般に，オペレーティング・システムの特定のインスタンスで使われるリソースは，構成ツリー中でそのパーティションを表すソフト・パーティションが所有します。 1 つのハード・パーティション内の複数のインスタンスで協調的なプッシュ・モデル使うことで，あるインスタンスが所有しているリソースは奪われることはなく，譲り渡すことだけが可能となります。ツリー上で上にあるノードが所有するリソースは，協調的に使用したり (たとえば，コミュニティ・レベルでの共用メモリ)，ツリーを下にたどって特定のソフト・パーティションに割り当てたり，利用可能になるまで上にたどって複数のソフト・パーティションに割り当てることができます。

ソフト・パーティション間でリソースを直接割り当てることを移動と呼びます。ソフト・パーティション間で移動できるのは CPU だけです。 パーティショニングでの移動機能は Galaxy 固有の機能ではありません。CPU は，あるソフト・パーティションから同じハード・パーティション内の他のソフト・パーティションに，ソフト・パーティション間で直接通信を行うことなく，移動できます。 CPU リソースの管理は，すべて OpenVMS の DCL コマンド `SET/STOP CPU` で行います。

注意:

ES47/ES80/GS1280 システムでの移動に関する制限事項:
IO が接続されている CPU を移動することはできません。 IO が接続されているか確認するには，実際の構成で CPU に接続されているケーブルを確認するか，MBM の show partition の出力で判断します。 この出力の IOPs セクションで，IO が接続されている CPU は破線で示されます。 CPU ID は，show partition の出力の CPUs セクション内の対応する PID です (1.3.1 項「ハード・パーティションの構成例」 の例を参照してください)。

図 1-4 ソフトウェア構成のツリー



1.3 AlphaServer ES47/ES80/GS1280 でのパーティショニング

1.3.1 ハード・パーティションの構成例

以下の例では，ES47/ES80/GS1280 システムでのハード・パーティションのセットアップを示します。 GCM (Graphical Configuration Manager) を使って，パーティション分割された構成の表示と管理を行う方法については，本書のGCMの章を参照してください。ソフト・パーティションのセットアップについては，第10章 「AlphaServer ES47/ES80/GS1280 システムでの OpenVMS Galaxy の構築」を参照してください。

次の例では，32 基のプロセッサを搭載したシステムで，3 つのハード・パーティションをセットアップしています。 パーティションのうち 2 つは，それぞれ 8 基のプロセッサを持ち，もう 1 つのパーティションは 16 基のプロセッサを持っています。 パーティションは 8P SBB 境界に割り当てられます。

サーバ管理のコマンド行インタフェース (CLI) については，次の場所にあるリファレンス・マニュアルを参照してください。

http://mediadocs.mro.cpqcorp.net/doc_library/GS1280/ServerMgnt/CLI_Reference.pdf

ハード・パーティションとソフト・パーティションの名前は，最大で 19 文字までです (英数字と下線のみ)。

注意:

サーバ管理の CLI では，大文字と小文字が区別されます。

この例では，コンソールは telnet ポート 323，324，および 325 でアクセスできます。 ハード・パーティションは part0，part1，および part2 という名前にし，それぞれデフォルトで 255 基の CPU で作成します。 サブパーティションの種別は soft (ソフト・パーティション) です。 その後，CPU と IO リソースを各パーティションに割り当てます。 `assign component`

コマンドでは，SBB 全体をコンポーネントに割り当てます。この例では，キャビネット内の各ドローワには 8 基のプロセッサが搭載されています。

これらコマンドに対する変更点については，前述の CLI_Reference.pdf マニュアルを参照してください。

```
Welcome - GS1280 Server Manager - T2.1-5
MBM> create partition -hp part0 255 soft
MBM> create partition -hp part1 255 soft
MBM> create partition -hp part2 255 soft
MBM> assign component -hp part0 -cabinet 0 -drawer 0 sbb
MBM> assign component -hp part1 -cabinet 0 -drawer 1 sbb
MBM> assign component -hp part2 -cabinet 0 -drawer 2 sbb
MBM> assign component -hp part2 -cabinet 0 -drawer 3 sbb
MBM> show partition
------------------------------------------------------------------


Hard Partition : HP Name = part0, HP No.= 0, SP count = 2
Attributes     : max CPUs = 16, SP type = soft, Non-stripe
Physical Memory: 16384MB (16.000GB)
 Community Memory: 0MB (0.000GB)
 Sub Partition: HP Name = part0, HP No.= 0
                SP Name = Default_SP, SP No.= 0
                State = Not Running, Telnet port = 323
 Assigned Memory: unspecified

 CPUs:

     Cab  Drw  CPU   (NS,EW)   PID    Type
      0    0    0    ( 0,0 )     0    Non-primary
      0    0    2    ( 0,1 )     2    Non-primary
      0    0    4    ( 0,2 )     4    Non-primary
      0    0    6    ( 0,3 )     6    Non-primary
      0    0    1    ( 1,0 )     1    Non-primary
      0    0    3    ( 1,1 )     3    Non-primary
      0    0    5    ( 1,2 )     5    Non-primary
      0    0    7    ( 1,3 )     7    Non-primary
 IOPs:

             SBB                            PCI Drawer
     Cab  Drw  IOP   (NS,EW)            Cab  Drw  IOR
      0    0    0    ( 0,0 ) ---------  2    4    0
      0    0    2    ( 0,1 )
      0    0    4    ( 0,2 )
      0    0    6    ( 0,3 )
      0    0    1    ( 1,0 )
      0    0    3    ( 1,1 )
      0    0    5    ( 1,2 )
      0    0    7    ( 1,3 )
 Sub Partition: HP Name = part0, HP No.= 0
                SP Name = Free_Pool, SP No.= 255
                State = Not Running
  Free Memory: 0MB (0.000GB)
  CPUs: None
  IOPs: None
------------------------------------------------------------------
Hard Partition : HP Name = part1, HP No.= 1, SP count = 2
Attributes     : max CPUs = 16, SP type = soft, Non-stripe
Physical Memory: 16384MB (16.000GB)

 Community Memory: 0MB (0.000GB)
 Sub Partition: HP Name = part1, HP No.= 1
```

```
                SP Name = Default_SP, SP No. = 0
                 State = Not Running, Telnet port = 324

 Assigned Memory: unspecified

 CPUs:

     Cab  Drw  CPU   (NS,EW)   PID    Type
      0    0    0    ( 2,0 )    0    Non-primary
      0    0    2    ( 2,1 )    2    Non-primary
      0    0    4    ( 2,2 )    4    Non-primary
      0    0    6    ( 2,3 )    6    Non-primary
      0    0    1    ( 3,0 )    1    Non-primary
      0    0    3    ( 3,1 )    3    Non-primary
      0    0    5    ( 3,2 )    5    Non-primary
      0    0    7    ( 3,3 )    7    Non-primary
 IOPs:

            SBB                            PCI Drawer
     Cab  Drw  IOP   (NS,EW)            Cab  Drw  IOR
      0    1    0    ( 2,0 ) ---------  2    5    0
      0    1    2    ( 2,1 )
      0    1    4    ( 2,2 )
      0    1    6    ( 2,3 )
      0    1    1    ( 3,0 )
      0    1    3    ( 3,1 )
      0    1    5    ( 3,2 )
      0    1    7    ( 3,3 )

 Sub Partition: HP Name = part1, HP No.= 1
                SP Name = Free_Pool, SP No. = 255
                State = Not Running
 Free Memory: 0MB (0.000GB)
 CPUs: None
 IOPs: None
-----------------------------------------------------------------------

Hard Partition : HP Name = part2, HP No.= 2, SP count = 2
Attributes     : max CPUs = 16, SP type = soft, Non-stripe
Physical Memory: 36864MB (36.000GB)
 Community Memory: 0MB (0.000GB)
 Sub Partition: HP Name = part2, HP No.= 2
                SP Name = Default_SP, SP No.= 0
                State = Not Running, Telnet port = 325
 Assigned Memory: unspecified
 CPUs:
     Cab  Drw  CPU   (NS,EW)   PID    Type
      0    2    0    ( 0,4 )    0    Non-primary
      0    2    2    ( 0,5 )    2    Non-primary
      0    2    4    ( 0,6 )    4    Non-primary
      0    2    6    ( 0,7 )    6    Non-primary
      0    2    1    ( 1,4 )    1    Non-primary
      0    2    3    ( 1,5 )    3    Non-primary
      0    2    5    ( 1,6 )    5    Non-primary
      0    2    7    ( 1,7 )    7    Non-primary
      0    3    0    ( 2,4 )    8    Non-primary
      0    3    2    ( 2,5 )   10    Non-primary
      0    3    4    ( 2,6 )   12    Non-primary
      0    3    6    ( 2,7 )   14    Non-primary
      0    3    1    ( 3,4 )    9    Non-primary
      0    3    3    ( 3,5 )   11    Non-primary
      0    3    5    ( 3,6 )   13    Non-primary
      0    3    7    ( 3,7 )   15    Non-primary
```

```
 IOPs:
            SBB                              PCI Drawer
    Cab  Drw  IOP    (NS,EW)            Cab  Drw  IOR
     0    2    0    ( 0,4 ) ---------  2    6    0
     0    2    2    ( 0,5 )
     0    2    4    ( 0,6 )
     0    2    6    ( 0,7 )
     0    2    1    ( 1,4 )
     0    2    3    ( 1,5 )
     0    2    5    ( 1,6 )
     0    2    7    ( 1,7 )
     0    3    0    ( 2,4 ) ---------  2    7    0
     0    3    2    ( 2,5 )
     0    3    4    ( 2,6 )
     0    3    6    ( 2,7 )
     0    3    1    ( 3,4 )
     0    3    3    ( 3,5 )
     0    3    5    ( 3,6 )
     0    3    7    ( 3,7 )
Sub Partition: HP Name = part2, HP No.= 2
               SP Name = Free_Pool, SP No.= 255
               State = Not Running
 Free Memory: 0MB (0.000GB)
 CPUs: None
 IOPs: None
MBM> save partition
```

この後システムの電源が投入され，診断が実行されます。

```
MBM> power on -all
[2003/04/16 08:05:31]
~PCO-I-(pco_04) Powering on partition. HP: part0
[2003/04/16 08:05:32]
~PCO-I-(pco_03) Powering on partition. HP: part1
[2003/04/16 08:05:32]
~PCO-I-(pco_01) Powering on partition. HP: part2
[2003/04/16 08:05:51]
~PCO-I-(pco_04) Running diagnostics on HP: part0
[2003/04/16 08:05:55]
~PCO-I-(pco_03) Running diagnostics on HP: part1
[2003/04/16 08:06:00]
~PCO-I-(pco_01) Running diagnostics on HP: part2
[2003/04/16 08:06:50]
~PCO-I-(pco_04) Diagnostics completed on HP: part0
[2003/04/16 08:06:50]
~PCO-I-(pco_04) HP:part0 SP:Default_SP Primary is NS:0 EW:0
 which is cab:00 drw:0 cpu:0 [2003/04/16 08:06:51]
~PCO-I-(pco_04) Loading SRM on Primary for HP: part0,
 SP: Default_SP. [2003/04/16 08:06:54]
~PCO-I-(pco_03) Diagnostics completed on HP: part1
2003/04/16 08:06:54]
~PCO-I-(pco_03) HP:part1 SP:Default_SP Primary is NS:2 EW:0
 which is cab:00 drw:1 cpu:0 [2003/04/16 08:06:54]
~PCO-I-(pco_03) Loading SRM on Primary for HP: part1,
 SP: Default_SP. [2003/04/16 08:06:55]
~PCO-I-(pco_04) Powered On HP:part0
[2003/04/16 08:06:59]
~PCO-I-(pco_03) Powered On HP:part1
[2003/04/16 08:07:24]
~PCO-I-(pco_01) Diagnostics completed on HP: part2
[2003/04/16 08:07:25]
~PCO-I-(pco_01) HP:part2 SP:Default_SP Primary is NS:0 EW:4
 which is cab:00 drw:2 cpu:0 [2003/04/16 08:07:25]
~PCO-I-(pco_01) Loading SRM on Primary for HP: part2,
 SP: Default_SP. [2003/04/16 08:07:29]
```

```
~PCO-I-(pco_01) Powered On HP:part2
MBM>
```

1.4 AlphaServer GS80/160/320 でのパーティショニング

ハード・パーティションは，次の要素を必要とします。

- パーティションごとの標準 COM1 UART コンソール・ライン
- パーティションごとの PCI ドローワ
- パーティションごとに 1 つの I/O モジュール
- パーティションごとに少なくとも 1 つの CPU モジュール
- パーティションごとに少なくとも 1 つのメモリ・モジュール

  メモリ・オプションは，メモリ帯域幅とメモリ容量に対するアプリケーションの感度，およびハードウェア・パーティションの数に従って選択してください。これによって必要なメモリ・ベース・モジュールとアップグレードの数が決まります。必要な合計容量は，選択される配列のサイズを決定します。

  メモリ・モジュールは，2 の倍数で構成してください。つまり，QBB のベース・モジュールの個数は，0，1，2，4 のいずれかにします。同様に，アップグレードでも QBB のベース・モジュールの個数は 2 の 倍数である 0，1，2，4 をインストールしてください。

これ以降の項では，次の 3 つのハード・パーティション構成の例を説明します。

- 構成例 1 には 4 つの QBB と 4 つのハード・パーティションがあります。それぞれのハード・パーティションには 1 つの QBB があります。
- 構成例 2 には 4 つの QBB と 2 つのハード・パーティションがあります。それぞれのハード・パーティションには 2 つの QBB があります。
- 構成例 3 には 4 つの QBB と 2 つのハード・パーティションがあります。1 つのパーティションには 1 つの QBB があり， もう 1 つのパーティションには 3 つの QBB があります。

ここで説明したパーティショニング手順の詳細については，『AlphaServer GS80/160/320 Firmware Reference Manual』 を参照してください。

1.4.1 ハード・パーティションの構成例 1

図 1-5 「構成例 1」は，4 つの QBB を使って 4 つのハード・パーティションを構成するものです。 それぞれのハード・パーティションには 1 つの QBB があります。

図 1-5 構成例 1

NODE	HP	QBB
WILD7	0	0
WILD8	1	1
WILD9	2	2
WILD10	3	3

VM-1061A-AI

2 つのハード・パーティションで AlphaServer GS160 システムを構成するには， 次の一連の SCM コマンドを入力します。

SCM コンソールから，hp NVRAM 変数の次の設定を入力します。 値はビット・マスクです。

```
SCM_E0> power off -all

SCM_E0> set hp_count 4
SCM_E0> set hp_qbb_mask0 1
SCM_E0> set hp_qbb_mask1 2
SCM_E0> set hp_qbb_mask2 4
```

```
SCM_E0> set hp_qbb_mask3 8
SCM_E0> set hp_qbb_mask4 0
SCM_E0> set hp_qbb_mask5 0
SCM_E0> set hp_qbb_mask6 0
SCM_E0> set hp_qbb_mask7 0

SCM_E0> power on -all
```

また，ハード・パーティションは電源を個別にオンまたはオフにできます。たとえば，この構成を使って，次のように入力することもできます。

```
SCM_E0> power off -all
SCM_E0> set hp_count 4
SCM_E0> set hp_qbb_mask0 1
SCM_E0> set hp_qbb_mask1 2
SCM_E0> set hp_qbb_mask2 4
SCM_E0> set hp_qbb_mask3 8
SCM_E0> set hp_qbb_mask4 0
SCM_E0> set hp_qbb_mask5 0
SCM_E0> set hp_qbb_mask6 0
SCM_E0> set hp_qbb_mask7 0
SCM_E0> power on -partition 0
SCM_E0> power on -partition 1
SCM_E0> power on -partition 2
SCM_E0> power on -partition 3
```

各ハード・パーティションの電源投入フェーズ中に，パーティションがどのようにオンラインになるかを示す状態情報が表示されます。この情報をよく観察し，処理中に障害が何も発生しないことを確認してください。

ハード・パーティションがオンラインになると，そのハード・パーティションのコンソール・デバイスでの作業を開始できます。 また，NVRAM 変数 AUTO_QUIT_SCM の設定に応じて，それぞれのハード・パーティションのコンソールが，SCM または SRM のどちらのコンソール・モードでもオンラインになることに注意してください。

ハード・パーティションのそれぞれのコンソールから SRM コンソールに入ると， そのハード・パーティションに固有のコンソール変数を設定できます。 その後，標準の OpenVMS の手順に従って， それぞれのハード・パーティションで OpenVMS をブートします。 次の例を参照してください。

OpenVMS における典型的なハード・パーティション 0 SRM コンソールの設定:

```
P00>>>show bootdef_dev
bootdef_dev             dkb0.0.0.3.0
P00>>>show boot_osflags
boot_osflags            0,0
P00>>>show os_type
os_type                 OpenVMS
```

OpenVMS における典型的なハード・パーティション 1 SRM コンソールの設定:

```
P00>>>show bootdef_dev
bootdef_dev             dkb0.0.0.3.0
P00>>>show boot_osflags
boot_osflags            1,0
P00>>>show os_type
os_type                 OpenVMS
```

OpenVMS における典型的なハード・パーティション 2 SRM コンソールの設定:

```
P00>>>show bootdef_dev
bootdef_dev             dkb0.0.0.3.0
P00>>>show boot_osflags
boot_osflags            2,1
```

```
P00>>>show os_type
os_type                 OpenVMS
```

Tru64 UNIX における典型的なハード・パーティション 3 SRM コンソールの設定:

```
P00>>>show bootdef_dev
bootdef_dev             dka0.0.0.1.16
P00>>>show boot_osflags
boot_osflags            A
P00>>>show os_type
os_type                 UNIX
```

1.4.2 ハード・パーティションの構成例 2

図 1-6 「構成例 2」 は，4 つの QBB を使って 2 つのハード・パーティションを構成するものです。それぞれのハード・パーティションには 2 つの QBB があります。

図 1-6 構成例 2

NODE	HP	QBB
WILD7	0	0, 1
WILD8	1	2, 3

VM-1063A-AI

2 つのハード・パーティションで AlphaServer GS160 を構成するには，次の一連の SCM コマンドを実行します。

SCM コンソールから，hp NVRAM 変数の次の設定を入力します。

```
SCM_E0> power off -all

SCM_E0> set hp_count 2
SCM_E0> set hp_qbb_mask0 3
SCM_E0> set hp_qbb_mask1 c
SCM_E0> set hp_qbb_mask2 0
SCM_E0> set hp_qbb_mask3 0
SCM_E0> set hp_qbb_mask4 0
SCM_E0> set hp_qbb_mask5 0
SCM_E0> set hp_qbb_mask6 0
SCM_E0> set hp_qbb_mask7 0

SCM_E0> power on -all
```

ハード・パーティションがオンラインになると，そのハード・パーティションのコンソール・デバイスでの作業を開始できます。

それぞれのハード・パーティションのコンソールから，構成例 1 と同様の手順で SRM コンソールに入り，そのハード・パーティションに固有のコンソール変数を構成してから，それぞれのハード・パーティションで OpenVMS をブートします。

1.4.3 ハード・パーティションの構成例 3

構成例 2 と同様，図 1-7 「構成例 3」 は 4 つの QBB を使って 2 つのハード・パーティションを構成するものです。唯一の違いは，それぞれのハード・パーティションの QBB の数が異なることです。

図 1-7 構成例 3

NODE	HP	QBB
WILD7	0	0, 1, 2
WILD8	1	3

VM-1062A-AI

AlphaServer GS160 システムを構成するには，次の一連の SCM コマンドを実行します。SCM コンソールから，hp NVRAM 変数の次の設定を入力します。

```
SCM_E0> power off -all

SCM_E0> set hp_count 2
SCM_E0> set hp_qbb_mask0 7
SCM_E0> set hp_qbb_mask1 8
SCM_E0> set hp_qbb_mask2 0
SCM_E0> set hp_qbb_mask3 0
SCM_E0> set hp_qbb_mask4 0
SCM_E0> set hp_qbb_mask5 0
SCM_E0> set hp_qbb_mask6 0
SCM_E0> set hp_qbb_mask7 0

SCM_E0> power on -all
```

他の例と同様，ハード・パーティションがオンラインになると，そのハード・パーティションのコンソール・デバイスでの作業を開始できます。

それぞれのハード・パーティションのコンソールから， 構成例 1 と同様の手順で SRM コンソールに入り，そのハード・パーティション固有の変数を構成して，それぞれのハード・パーティションで OpenVMS をブートします。

1.4.4 AlphaServer GS80/160/320 システムでのコンソール・ファームウェアの更新

ハード・パーティション分割されたシステムで SRM コンソール・ファームウェアを更新するには，それぞれのハード・パーティションに対して個別に更新を行う必要があります。各パーティション上の全ファームウェアを一度にアップデートする方法はありません。

1.5 OpenVMS Galaxy サポート

OpenVMS Galaxy ソフトウェアは，共用メモリを通してオペレーティング・システムのインスタンスを制御し，複数のソフト・パーティション内のインスタンス間でリソース共有関係を実現します。 複数の独立したオペレーティング・システムのインスタンスは，Galaxy なしでも複数のソフト・パーティションで動作できます。OpenVMS Galaxy の概念についての詳細は，第2章 「OpenVMS Galaxy の概念」を参照してください。 複数のソフト・パーティションを作成するには，使用するハードウェアに応じて，第 4 章 ～ 第 10 章に示す Galaxy 関連の手順に従ってください。

構成ツリーにより，各ハード・パーティション内でツリーの上下にリソースを操作できるようになります。 ツリーは物理的な接続とリソースの所有関係を定義します。 ハードウェア・リソースには，いくつもあるレベルのどれかのレベルで所有関係を割り当てることができますが，リソースを割り当てて使用するのは 1 つのインスタンスです。 インスタンスとは，ソフト・パーティション内で動作する 1 つのオペレーティング・システムのことです。

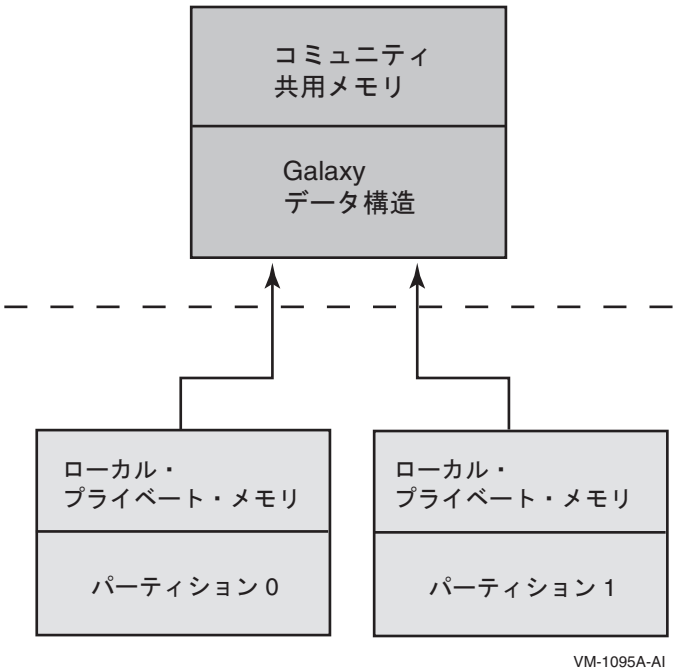
システム・リソースはソフト・パーティションが管理します。 CPU はソフト・パーティションによって所有されている場合にのみ使用され，そのパーティションで動作するインスタンスにより操作されます。 ただし，CPU は構成ツリーの上位レベルが所有することもできます。この場合，どのインスタンスからも平等に利用できるようになります。 プラットフォーム・

ボックスに電源を投入する前にブートの所有権が設定されておらず，CPU モジュールが後からシステムに追加された場合には，その CPU は追加されたハード・パーティションが所有することになり，そのハード・パーティション内の任意のソフト・パーティションにプログラム的に割り当てることができます (ES47/ES80/GS1280 では，コンポーネントのホット・スワップはサポートされません)。

オペレーティング・システムのインスタンスのどれかがブートするとすぐに，そのインスタンスが割り当てられているソフト・パーティションがインスタンスのすべてのリソースを排他的に所有し，その状態特性はそのインスタンスだけが操作できるようになります。そのため，たとえ今は CPU がなかったとしても，使うことになった場合に備え，電源投入時の CPU の初期割り当てを検討し，リソースが最適に配分されるようにしておくのは重要なことです。

1 つのハード・パーティション内に複数のソフト・パーティションを作成するには，前述のように標準的なパーティショニング手順を使用して，各ソフト・パーティションでインスタンスが動作するような Galaxy 構成を作成します。 図 1-8「メモリの使われ方」 は，Galaxy インスタンス内でどのようにメモリが使われるかを示します。各ソフト・パーティションには，最低限 OpenVMS 用のメモリが必要です。

図 1-8 メモリの使われ方



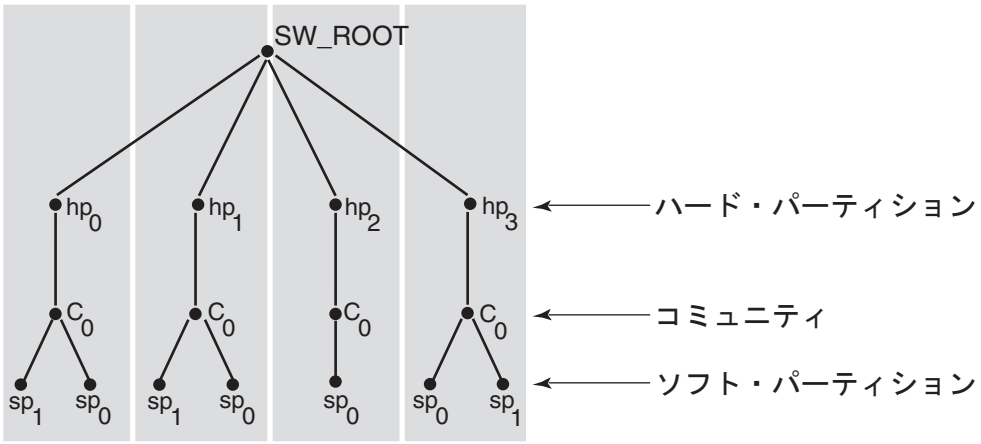
Galaxy ID はハード・パーティションの内部にあり，そのハード・パーティションの範囲で同じになります。 つまり，たとえば，2 つのハード・パーティションがあって，両方で Galaxy を動かすと，それぞれの Galaxy が固有の Galaxy ID を持つことになります。 ネットワーク管理ツールを使うときにはこれを考慮してください。 2 つの Galaxy ID があると，ネットワーク管理ツールは 2 つの Galaxy 環境を参照します。

図 1-9「構成ツリーで表したソフト・パーティショニング」 では，4 つのハード・パーティション (0, 1, 2, 3) を示しており，各パーティションには hp_0 のような一意の名前がついています。 ハード・パーティション 0 には，一番左の sp_1 と sp_0 の 2 つのソフト・パーティションを含む Galaxy があります。 他の各ハード・パーティションにも Galaxy があります。ハード・パーティションで Galaxy が作成されておらず，Galaxy に参加もしていない場合には，複数の独立したオペレーティング・システムのインスタンスができることになります。どのソフト・パーティションでも，オペレーティング・システムのインスタンスを 1 つ動作させることができます。 Galaxy が使用する共用メモリはコミュニティが所有しているため，コミュニティごとに Galaxy が 1 つ存在します。 これにより，Galaxy に属するすべてのインスタンスから共用メモリを参照したりアクセスしたりできるようになります。コミュニティ・ノードの下のソフト・パーティションで動作するすべてのインスタンスは，その Galaxy に参加する資

格があります。メンバ・ノードは Galaxy によって制御される共有リソースを利用できます。独立したインスタンスも CPU の割り当てと移動はできますが，共用メモリを利用できるのは Galaxy のメンバだけです。

コミュニティは，ハード・パーティションごとに 1 つしか存在できません。

図 1-9 構成ツリーで表したソフト・パーティショニング



VM-1099A-AI

1.6 RAD (リソース・アフィニティ・ドメイン) に対する OpenVMS アプリケーション・サポート

新しい AlphaServer GS シリーズ・システムには大量の物理メモリがあるので， 非常に大きなデータベースを完全にメモリ内に入れることができます。 AlphaServer の **NUMA (ノンユニフォーム・メモリ・アクセス)** システム・アーキテクチャが， この大量のメモリを効率的にアクセスするための帯域幅を提供します。NUMA は， 物理メモリへのアクセス時間がすべての CPU で同じではないシステムの属性です。

OpenVMS エンジニアリング・グループは，OpenVMS Alpha バージョン 7.2–1H1 で，OpenVMS メモリ管理とプロセス・スケジューリングを NUMA 対応にしました。この機能 (RAD へのアプリケーション・サポート) によって， 複数のビルディング・ブロック上の単一の OpenVMS インスタンスで実行されるアプリケーションは NUMA 環境でできるだけ効率的に実行できるようになります。

オペレーティング・システムは，ハードウェアを RAD (リソース・アフィニティ・ドメイン) 群として扱います。 RAD とは，共通のアクセス特性を持つ一連のハードウェア・コンポーネント (CPU，メモリ，I/O) のことです。AlphaServer GS80/160/320 システムでは， RAD は QBB (quad building block) に相当します。 AlphaServer ES47/ES80/GS1280 システムでは，RAD は 2 プロセッサの CPU ボードに相当します。 CPU が同一の RAD 内のメモリを参照する速度は， 別の RAD 内のメモリを参照する速度より最大で約 3 倍速くなります。 このため，一部のプロセスにのみ常に利点を偏って提供しないように，コードの実行とメモリの参照をできるだけ同一の RAD 内に留めることが重要になります。良い配置が良い性能の鍵になりますが，公平さが重要なときにはできるだけ公平にしなければなりません。

OpenVMS スケジューラとメモリ管理サブシステムは，ともに次のようにして可能な限り最高の配置を行います。

- 各プロセスに優先 RAD または " ホーム " RAD を割り当てる
- ホーム RAD のメモリからプロセスのプライベート・ページを割り当てる
- 通常はホーム RAD 内の CPU にプロセスをスケジューリングする
- 各 RAD にオペレーティング・システムの読み込み専用コードを複製する
- RAD にグローバル・ページを配布する
- RAD に予約メモリをストライピングする

OpenVMS RAD アプリケーション・プログラミング・インタフェースの 使用方法についての詳細は，第3章「OpenVMS アプリケーションに対する NUMA の影響」を参照してください。

第2章 OpenVMS Galaxy の概念

OpenVMS Alpha で HP Galaxy ソフトウェア・アーキテクチャを使用すると，1 台のコンピュータで OpenVMS の複数のインスタンスを実行できます。 システム・リソースは動的に再割り当てすることができ，コンピュータをリブートしなくても，必要に応じてコンピュータ・パワーをアプリケーションに割り当てることができます。

この章では，OpenVMS Galaxy の概念について， OpenVMS Alpha バージョン 7.3 で提供される機能を中心に説明します。

2.1 OpenVMS Galaxy の概念およびコンポーネント

ソフトウェアは CPU，メモリ，I/O ポートを OpenVMS オペレーティング・システムの個々のインスタンスに割り当てることで，これらのリソースを論理的に **パーティション分割** します。このパーティション化はシステム管理者が行うものであり，ソフトウェアの機能です。ハードウェアを物理的に分割する必要はありません。各インスタンスには，動作に必要なリソースが割り当てられます。OpenVMS Galaxy 環境は， CPU などのリソースを OpenVMS の異なるインスタンスに動的に再割り当てすることができるという点で，**適応型** です。

OpenVMS Galaxy ソフトウェア・アーキテクチャには， 次のハードウェアおよびソフトウェア・コンポーネントが含まれています。

コンソール

OpenVMS システムの **コンソール** は，接続されている端末と， ファームウェア・プログラムで構成されます。 ファームウェア・プログラムは電源投入時の自己診断テスト， ハードウェアの初期化，システム・ブートの開始， システム・ブートとシャットダウン時の I/O サービスを実行します。 また，コンソール・プログラムはコンソール端末の入出力，環境変数の検索， NVRAM (不揮発性ランダム・アクセス・メモリ) の保存をはじめ， その他のさまざまなサービスのために， オペレーティング・システムに実行時サービスも提供します。

OpenVMS Galaxy コンピューティング環境では， コンソールはハードウェア・リソースのパーティション分割で重要な役割を演じます。NVRAM に永久的なシステム構成を保存し，稼働中の構成をメモリに記憶します。コンソールは，OpenVMS オペレーティング・システムの各インスタンスに対して， 実行中の構成データを指すポインタを与えます。

共用メモリ

メモリは論理的に，プライベート・セクションと共用セクションに分割されます。 オペレーティング・システムの各インスタンスには， それぞれ独自のプライベート・メモリがあります。つまり，他のインスタンスはプライベート・メモリの物理ページをマッピングしません。共用メモリの一部は，OpenVMS のインスタンスが他のインスタンスと通信するために使用され， 残りの共用メモリはアプリケーション用に使用されます。

Galaxy ソフトウェア・アーキテクチャは， ノンユニフォーム・メモリ・アクセス (NUMA) 環境用に準備されており，このようなシステムが最大限のアプリケーション性能を実現できるように， 必要に応じて特殊なサービスを提供します。

CPU

OpenVMS Galaxy コンピューティング環境では， インスタンス間で CPU を再割り当てすることができます。

I/O

OpenVMS Galaxy には，スケーラビリティの高い I/O サブシステムがあります。 これは，各インスタンスに対して 1 つずつ， 複数のプライマリ CPU がシステム内にあるからです。 また，OpenVMS は現在， SMP システムのセカンダリ CPU に一部の I/O を分散するための機能も備えています。

独立インスタンス

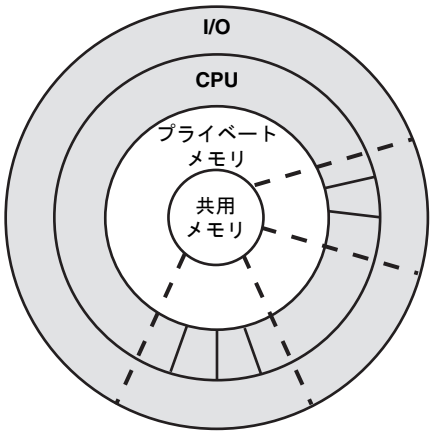
OpenVMS Galaxy では，リソースをまったく共用せずに，1 つ以上の OpenVMS インスタンスを実行できます。リソースを共用しない OpenVMS インスタンスを **独立インスタンス** と呼びます。

OpenVMS の独立インスタンスは共用メモリの使用に参加しません。基本オペレーティング・システムもアプリケーションも，共用メモリにアクセスしません。

OpenVMS Galaxy は独立インスタンスだけで構成することができます。このようなシステムは従来のメインフレーム・スタイルのパーティション化によく似ています。アーキテクチャの面から考えると，OpenVMS Galaxy は 対称型マルチプロセッサ (SMP) ハードウェア・アーキテクチャを基礎にしています。CPU，メモリ，I/O はマシン内で完全に接続されているものと想定されており，メモリはキャッシュと密接に結合されているものと想定されています。各サブシステムは他のすべてのサブシステムに完全にアクセスできます。

図 2-1 「OpenVMS Galaxy アーキテクチャの概念図」 に示すように，Galaxy ソフトウェアはリソースをパイであると考えます。さまざまなリソース (CPU，プライベート・メモリ，共用メモリ，I/O) は，特定の階層構造でパイの内部に同心円として並べられます。その場合，共用メモリが中心になります。

図 2-1 OpenVMS Galaxy アーキテクチャの概念図

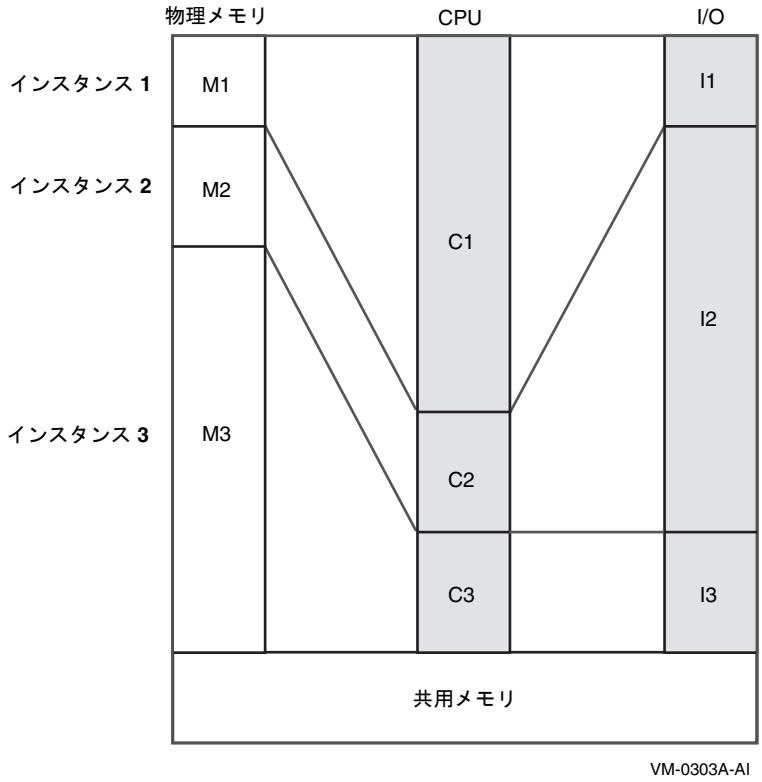


Galaxy では，それぞれ異なるサイズの複数のスライスにパイを分割する機能がサポートされます。各スライスは，サイズとは無関係に，すべての共用メモリにアクセスできます。さらに，パイを分割するのはソフトウェアであるため，スライスの数やサイズは動的に変化させることができます。

つまり，パイの各スライスはオペレーティング・システムの完全なインスタンスです。各インスタンスには専用のプライベート・メモリ，多くの CPU，必要な I/O が割り当てられます。各インスタンスは，アプリケーション・データが格納されているすべての共用メモリを確認できます。システム・リソースは，リブートせずにオペレーティング・システムのインスタンス間で再割り当てすることができます。

たとえば，図 2-2 「別の Galaxy アーキテクチャ・ダイアグラム」 は，Galaxy の比率がインスタンスによってどのように違うかを示しています。

図 2-2 別の Galaxy アーキテクチャ・ダイアグラム



2.2 OpenVMS Galaxy の機能

OpenVMS 機能の進歩にともない，OpenVMS Galaxy では，実績のある OpenVMS Cluster，対称型マルチプロセシング，性能を向上するためのさまざまな機能を活用して，これまでより高いレベルの性能，スケーラビリティ，可用性を提供しており，非常に柔軟な操作機能も提供します。

クラスタ

OpenVMS Cluster テクノロジには 15 年にわたる実績があり， OpenVMS Galaxy 内でクラスタ接続されたインスタンス間で通信を容易にします。

OpenVMS Cluster はソフトウェアの概念です。これは，***各コンピュータで 1 つずつ動作している***OpenVMS オペレーティング・システムを結合したものであり，さまざまな媒体を介して通信することで， 複数のコンピュータの処理パワーと記憶容量を 1 つの共用環境として構築します。

OpenVMS Galaxy もソフトウェア概念です。 しかし，これは *1 **台のコンピュータ*** で協調動作する OpenVMS オペレーティング・システムの集合であり， 共用メモリを通じて通信します。OpenVMS Galaxy では， オペレーティング・システムのインスタンスを Galaxy 内の他のインスタンスや他のシステムのインスタンスとクラスタ接続することができます。

OpenVMS Galaxy はそれ自体で完全なシステムです。 現在，クラスタにノードを追加できるのと同様に， OpenVMS Galaxy も既存の OpenVMS クラスタに追加することができますが，OpenVMS Galaxy アーキテクチャではシングル・システムという点に焦点を絞っています。OpenVMS Galaxy 内で完全に稼働するアプリケーションは， マルチシステム・クラスタでは実現できない高い性能を活用することができます。

SMP

OpenVMS Galaxy のインスタンスは SMP 構成にすることができます。 CPU の数はインスタンスの定義の一部です。OpenVMS Galaxy のインスタンスは， 完全な OpenVMS オペレーティング・システムであるため， すべてのアプリケーションが従来のシングル・インスタンス・コンピュータの場合と同様に動作します。

CPU の再割り当て

CPU はあるインスタンスから別のインスタンスに動的に再割り当てすることができ，その間も 2 つのインスタンス上ですべてのアプリケーションの実行を継続できます。再割り当ては 3 つの独立した機能によって実現されます。それは，再割り当てする CPU の停止，再割り当て，起動です。アプリケーションで必要なリソースが変化すると，CPU を適切なインスタンスに再割り当てすることができます。しかし，いくつかの制限があります。たとえば，インスタンスのプライマリ CPU を再割り当てすることができず，特定の割り込みを処理するために，特定の CPU を指定することはできません。

動的な再構成

OpenVMS オペレーティング・システムの複数のインスタンスを実行することで，システム管理者は，処理パワーを最も必要としているアプリケーションを稼働しているインスタンスにその処理パワーを再割り当てすることができます。要件の変化にともなって，構成も変更することができます。OpenVMS では，すべてのインスタンスおよびそのアプリケーションの実行を続行しながら，動的な再構成が可能です。

2.3 OpenVMS Galaxy の利点

OpenVMS Galaxy テクノロジの多くの利点は，OpenVMS オペレーティング・システムの複数のインスタンスを 1 台のコンピュータで実行することで実現されています。

OpenVMS の複数のインスタンスを同時にメモリに格納することで，OpenVMS Galaxy コンピューティング環境は次の機能を向上させます。

- 互換性

  既存のアプリケーションは，変更せずにそのまま実行できます。

- 可用性

  システムをダウンさせずに，ソフトウェアのアップグレードやシステム容量の拡張を行うことができます。

- スケーラビリティ

  SMP 環境やクラスタ環境の性能を向上するためのスケール機能を提供します。

- 適応性

  絶えず変化する作業負荷に対応するために，物理リソースを動的に割り当てることができます。

- 所有コスト

  必要なコンピュータ・システムの台数が少なくなるため，システム管理作業やシステムの設置空間を削減することができます。

- 性能

  多くのボトルネックが排除され，可能な I/O 構成の可能性が広がりました。

ここでは，これらの利点についてさらに詳しく説明します。

互換性

既存のシングル・システム・アプリケーションは，まったく変更せずに，OpenVMS Galaxy のインスタンスで実行できます。既存の OpenVMS Cluster アプリケーションも，OpenVMS Galaxy のクラスタ接続されたインスタンスで，変更せずに実行できます。

可用性

OpenVMS Galaxy システムは，従来のシングル・システム SMP システムより高い可用性を提供できます。これは，オペレーティング・システムの複数のインスタンスがハードウェア・リソースを制御するからです。

OpenVMS Galaxy では，OpenVMS の複数のバージョン (バージョン 7.2 以上) を同時に実行できます。たとえば，現在のバージョンを 1 つのインスタンスで実行しながら，オペレーティング・システムやアプリケーションの新しいバージョンを別のインスタンスでテストすること

ができます。 その後，一度に 1 つずつ， 各インスタンスでシステム全体をアップグレードすることができます。

スケーラビリティ

システム管理者は，ビジネス・ニーズの拡大や変化にともなって，アプリケーションの要件に対応するようにリソースを割り当てることができます。CPU が Galaxy 構成に追加された場合は， OpenVMS のどのインスタンスにでも割り当てることができます。 つまり，アプリケーションは CPU のパワーを 100 パーセント利用できるのです。

SMP でスケーラビリティに関して一般的に発生する問題点のために， OpenVMS Galaxy が制限を受けることはありません。 システム管理者は OpenVMS インスタンスの数を定義し， 各インスタンスで CPU の数を割り当て，その使用方法を制御することができます。

さらに，試行錯誤を繰り返しながらリソースを評価することができます。システム管理者は最も効果的なリソースの組み合わせが見つかるまで， OpenVMS のインスタンス間で CPU を再割り当てすることができます。 OpenVMS のすべてのインスタンスとそのアプリケーションは， CPU を再割り当てしている間も実行を継続できます。

適応性

OpenVMS Galaxy では，すべてのアプリケーションの実行を継続しながら， コンピューティング・リソースをオペレーティング・システムの他のインスタンスに動的に再割り当てすることができるので， 非常に高い適応性を実現できます。

CPU の再割り当て機能は，OpenVMS Galaxy コンピューティング環境の適応機能をもっとも顕著に表すものです。 たとえば，特定の時刻に必要とされるリソースが変化することがシステム管理者にわかっている場合は，システム管理者は CPU を OpenVMS の他のインスタンスに再割り当てするコマンド・プロシージャを作成し，そのプロシージャをバッチ・キューに登録することができます。 同様の方法で，システム負荷も管理することができます。

OpenVMS Galaxy 環境では，ハードウェア・リソースの割り当てや動的な再割り当ては， ソフトウェアによって完全に制御されます。 ハードウェアが OpenVMS Galaxy システムに追加されると，リソースを既存のインスタンスに追加することも，新しいインスタンスを定義することもできます。 その場合，実行中のアプリケーションに影響ありません。

所有コスト

OpenVMS Galaxy では，コンピュータがクラスタ・メンバの場合も， 独立システムの場合も，オペレーティング・システムの複数のインスタンスが 1 台のコンピュータで動作するので，既存のコンピュータをアップグレードし，処理能力を拡大することができ，また，一部のコンピュータを交換することもできます。 必要なコンピュータの台数が削減されるため， 必要なシステム管理作業や設置空間も大幅に削減されます。

性能

OpenVMS Galaxy では，SMP やクラスタ・テクノロジで発生する多くのスケール・ボトルネックが排除されることで， 事務処理アプリケーションの性能を向上できます。 また，インスタンス間で割り込みを分散することができるため， 多くの I/O 構成が可能になります。 たとえば，特定の I/O トラフィックが特定のインスタンスで実行されるように， システムの I/O 作業負荷を分割することができます。

2.4 OpenVMS Galaxy バージョン 7.3 の機能

OpenVMS Alpha バージョン 7.3 では，OpenVMS Galaxy 環境を構築して， 次の機能を実現できます。

- AlphaServer GS320 または GS1280/32 で 8 つのインスタンスを実行できます。
- AlphaServer GS160，ES80，GS1280/16 で 4 つのインスタンスを実行できます。
- AlphaServer GS160，ES47，GS1280/8 で 2 つのインスタンスを実行できます。
- AlphaServer GS140 または 8400 システムで OpenVMS の 3 つのインスタンスを実行できます。
- AlphaServer GS60，GS60E，GS80，8200，4100 または ES40 システムで OpenVMS の 2 つのインスタンスを実行できます。

- インスタンス間で CPU を再割り当てすることができます。
- インスタンスを個別にブートおよびシャットダウンできます。
- インスタンス間の通信のために共用メモリを使用できます。
- 共用メモリ・クラスタ・インターコネクトを使用して， OpenVMS Galaxy 内でインスタンスをクラスタ接続できます。
- Galaxy 以外のシステムとインスタンスをクラスタ接続できます。
- イベント通知，同期のためのロック，グローバル・セクション用の共用メモリ，リソース管理のための OpenVMS Galaxy API を使用してアプリケーションを開発できます。
- Galaxy Configuration Utility (GCU) を使用して， OpenVMS Galaxy 環境を表示し，制御することができます。(詳細については，第13章「OpenVMS Galaxy Configuration ユーティリティ」を参照してください。)
- アプリケーション開発のために，1 台の Alpha システムでシングル・インスタンス OpenVMS Galaxy を実行できます。

2.5 OpenVMS Galaxy の利点

IT 作業負荷が予測不可能な場合や大きく変動する場合，あるいは急激な増加を示している場合には， 作業負荷の管理能力を強化する必要があります。 このような場合，OpenVMS Galaxy テクノロジを導入することで，非常に柔軟な方法でシステム・リソースの動的な再構成と管理が可能になります。 OpenVMS Galaxy はハードウェアとソフトウェアを統合したソリューションであり， システム管理者は簡単なドラッグ・アンド・ドロップ操作で， 各 CPU の再割り当てなどの作業を実行できます。

OpenVMS Galaxy コンピューティング環境は， 次のような高可用性アプリケーションにとって理想的です。

- データベース・サーバ
- トランザクション処理システム
- データ・ウェアハウス
- データ・マイニング
- インターネット・サーバ
- NonStop eBusiness ソリューション

OpenVMS Galaxy コンピューティング環境は， クラスタや複数の独立したシステムをご利用の， 現在の OpenVMS ユーザのための自然な発展の道筋です。

OpenVMS Galaxy は，作業負荷が予測可能な場合も予測不可能な場合も， 成長の著しい組織にとって魅力的です。

2.6 OpenVMS Galaxy が最良の選択ではない場合

OpenVMS Galaxy テクノロジは非常に柔軟な方法でシステム・リソースの動的な再構成と管理を可能にしますが， Galaxy システムが組織にとって最良の技術的選択ではない場合もあります。 組織のコンピューティング・ニーズの焦点が次のようなものに向けられている場合には，OpenVMS Galaxy は最良の選択ではありません。

- シングル・ストリームの非スレッド化計算。
- プロセス制御などのために，個々の CPU が物理的に分離され， 他の機器の近くに配置される必要がある状況。
- 従来の SMP で効果がない場合。

2.7 さまざまな OpenVMS Galaxy 構成

OpenVMS Galaxy コンピューティング環境では， 1 台のコンピュータ・システムのインスタンス間でどの程度の協調動作を行うかを， お客様が判断できます。

shared-nothing コンピューティング・モデルでは，インスタンスはリソースを共用せず，操作は相互に分離されます (2.7.1 項「shared-nothing コンピューティング・モデル」を参照)。

shared-partial コンピューティング・モデルでは，インスタンスは一部のリソースを共用し，制限された方法で協調動作します (2.7.2 項「shared-partial コンピューティング・モデル」を参照)。

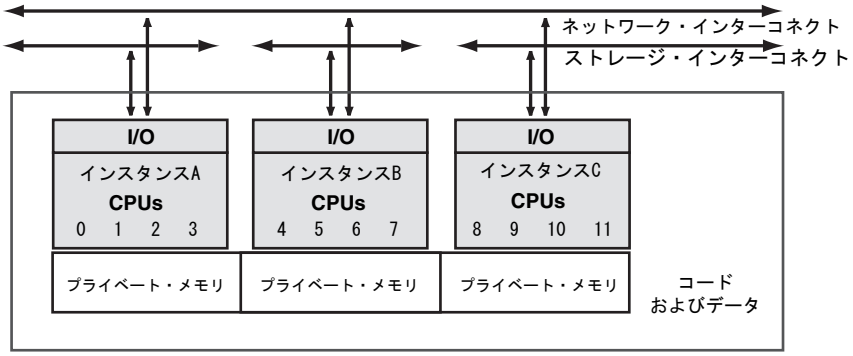
shared-everything モデルでは，インスタンスは完全に協調動作し，使用可能なすべてのリソースを共用します。この結果，オペレーティング・システムはネットワークにとって 1 つの結合されたものとして認識されます (2.7.3 項「shared-everything コンピューティング・モデル」を参照)。

2.7.1 shared-nothing コンピューティング・モデル

shared-nothing 構成 (図 2-3 「shared-nothing コンピューティング・モデル」 を参照) では，OpenVMS のインスタンスは相互に完全に独立しており，独立したコンピュータであるかのように，外部インターコネクトを通じて接続されます。

Galaxy では，使用可能なすべてのメモリが OpenVMS の各インスタンスのプライベート・メモリに割り当てられます。各インスタンスにはそれぞれ独自の CPU セットと適切な量の I/O リソースが割り当てられます。

図 2-3 shared-nothing コンピューティング・モデル

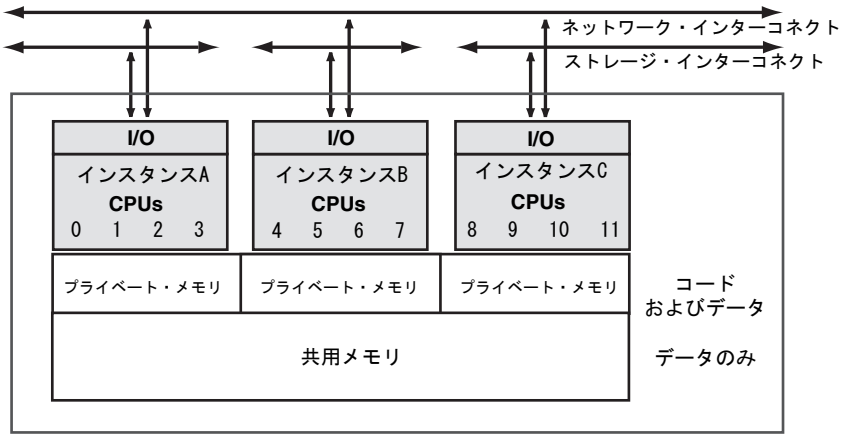


2.7.2 shared-partial コンピューティング・モデル

shared-partial 構成 (図 2-4 「shared-partial コンピューティング・モデル」 を参照) では，システム・メモリの一部が共用メモリとして指定され，各インスタンスがアクセスできます。各インスタンスのコードとデータはプライベート・メモリに格納されます。複数のインスタンスのアプリケーション間で共用されるデータは，共用メモリに格納されます。

インスタンスはクラスタ接続されません。

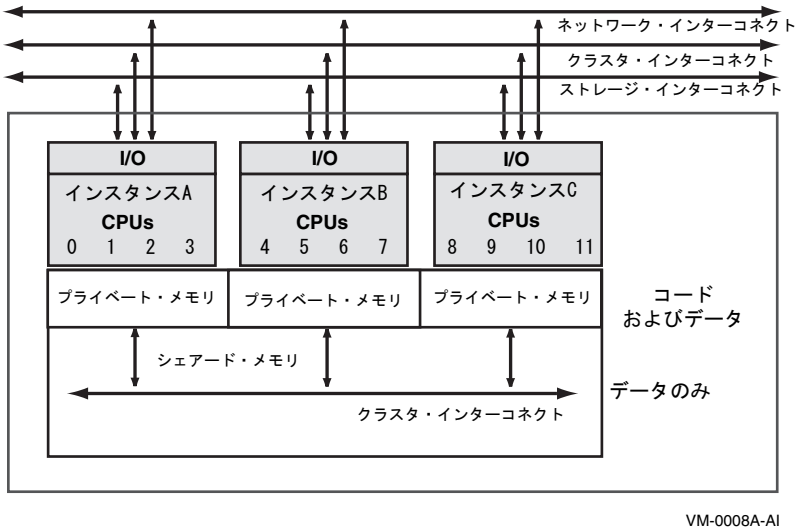
図 2-4 shared-partial コンピューティング・モデル



2.7.3 shared-everything コンピューティング・モデル

shared-everything 構成 (図 2-5 「shared-everything コンピューティング・モデル」 を参照) では，インスタンスはメモリを共用し，相互にクラスタ接続されます。

図 2-5 shared-everything コンピューティング・モデル



2.8 シングル・インスタンス Galaxy 構成

シングル・インスタンス Galaxy は，Galaxy コンソールのないシステムです。 通常コンソール・ファームウェアから提供される Galaxy 構成データは，ファイルに作成されます。システム・パラメータ GALAXY を 1 に設定することで，SYSBOOT はファイルをメモリに読み込み，システムはシングル・インスタンス Galaxy としてブートされます。 共用メモリ，Galaxy システム・サービス，さらに CPU のセルフ・マイグレーションも備えています。 この構成は，ES45 以外のどの Alpha プラットフォームでも可能です。

シングル・インスタンス Galaxy 構成は，ラップトップからメイン・フレームまでどのプラットフォームでも動作します。 この機能を利用すれば，早期に OpenVMS Galaxy 機能を評価することができ，さらにもっとも重要なこととして，完全なスケールの Galaxy プラットフォームを準備しなくても，Galaxy 対応アプリケーションを開発し，テストすることができるという利点があります。

シングル・インスタンス Galaxy はエミュレータではなく，実際の Galaxy コードであるため，アプリケーションはマルチインスタンス構成で動作します。

シングル・インスタンス Galaxy の実行の詳細については， 第11章 「Alpha システムでのシングル・インスタンス Galaxy の使用」 を参照してください。

2.9 OpenVMS Galaxy の構成上の考慮点

OpenVMS Galaxy コンピューティング環境を構築する場合は，構成にとって適切なハードウェアを用意する必要があります。 OpenVMS Galaxy の構成では，一般に次の規則が適用されます。

- インスタンスごとに 1 つ以上の CPU
- インスタンスごとに 1 つ以上の I/O モジュール
- インスタンスごとに専用のシリアル・コンソール・ポートまたはネットワーク・アクセス
- メモリ
  - OpenVMS とアプリケーション用に十分なプライベート・メモリ
  - 共用メモリ・クラスタ・インターコネクト，グローバル・セクションなどのために十分な共用メモリ

- Xterminal エミュレータを使用して DECwindows または Windows NT ワークステーションを稼働する Alpha または VAX ワークステーションでの構成管理情報の表示

各ハードウェア固有の構成の要件の詳細については，使用しているハードウェアに関する本書の章を参照してください。

2.9.1 XMI バスのサポート

XMI バスは AlphaServer 8400 システムで Galaxy 構成の最初のインスタンス (インスタンス 0) でのみサポートされます。

AlphaServer 8400 システムでは，すべての XMI 装置に対して DWLM-AA XMI プラグイン・ユニット・サブシステム・ケージは 1 つだけサポートされます。 すべての XMI 装置をシステムに接続するために， システムの背面に I/O バルクヘッドが必要なため，DWLM-AA はシステムである程度の空間を使用します。 このため， システム内に DWLPB PCI プラグイン・ユニットは 2 つだけしか追加できません。

2.9.2 メモリ分割に関する制限事項

下記のメモリ粒度の制限事項に注意してください。

- プライベート・メモリは 64 MB 境界から開始しなければなりません。
- 共用メモリは 8 MB 境界から開始しなければなりません。
- 最後のインスタンスを除き，他のすべてのインスタンスのメモリは 64 MB の倍数でなければなりません。

メモリを誤った方法で構成すると，一般にメモリが無駄になります。

2.9.3 EISA バスのサポート

EISA バスは Galaxy 構成の最初のインスタンス (インスタンス 0) でのみサポートされます。すべての EISA オプションの設計により，これらはシステムのインスタンス 0 に必ず存在しなければなりません。 Galaxy システムのどの EISA 装置の場合も，KFE70 は最初のインスタンスで使用しなければなりません。

すべての EISA 装置はインスタンス 0 に存在しなければなりません。 Galaxy システムの他のインスタンスでは，EISA 装置はサポートされません。

他のインスタンスにインストールされた KFE72-DA は，コンソール接続だけを提供し， 他の EISA 装置に対して使用することはできません。

2.10 推奨される CD ドライブ

OpenVMS Galaxy コンピューティング環境の各インスタンスに対して， CD ドライブを使用できるように設定してください。 OpenVMS Galaxy で複数のシステム・ディスクを使用する予定がある場合は， アップグレードやソフトウェアのインストール時に， 各インスタンスに 1 台ずつ CD ドライブがあると便利です。

OpenVMS Galaxy インスタンスが相互にクラスタ接続されており，1 つの共通のシステム・ディスクを使用する場合は，1 台の CD ドライブだけで十分です。 これは，他のクラスタ接続されたインスタンスに CD ドライブを提供できるからです。 オペレーティング・システムをアップグレードする場合は，CD ドライブが接続されているインスタンスを使用してアップグレードできます。

2.11 Galaxy を使ったクラスタ

ここでは，OpenVMS Galaxy コンピューティング環境内の他のインスタンスや，Galaxy 以外の OpenVMS クラスタと，インスタンスをクラスタ接続するための情報をまとめます。

クラスタ・インスタンスに適用される，必要な OpenVMS Galaxy ライセンスの詳細については，『OpenVMS License Management Utility Manual』を参照してください。

2.11.1 OpenVMS Galaxy インスタンスになる

OpenVMS Alpha バージョン 7.3 をインストールするときに，OpenVMS インストール・ダイアログで OpenVMS Cluster および OpenVMS Galaxy インスタンスに関する質問が表示されます。

次の質問に対して “Yes” と応答し，

```
Will this system be a member of an OpenVMS cluster? (Yes/No)
```

次の質問に対して “Yes” と応答すると，

```
Will this sytem be an instance in an OpenVMS Galaxy? (Yes/No)
```

次の情報が表示されます。

```
For compatibility with an OpenVMS Galaxy, any systems in the OpenVMS cluster
which are running versions of OpenVMS prior to V7.1-2 must have a
remedial kit installed.  The appropriate kit from the following list
must be installed on all system disks used by these systems.
(Later versions of these remedial kits may be used if available.)

Alpha V7.1 and V7.1-1xx         ALPSYSB02_071
Alpha V6.2 and V6.2-1xx         ALPSYSB02_062

VAX V7.1                        VAXSYSB01_071
VAX V6.2                        VAXSYSB01_062
```

詳細については，『OpenVMS インストレーション・ガイド [翻訳版]』を参照してください。

2.11.2 SCSI Cluster に関する留意事項

ここでは，OpenVMS Galaxy コンピューティング環境での SCSI 装置名に関する情報をまとめます。OpenVMS Cluster 装置名の詳細については，『OpenVMS Cluster システム』を参照してください。

共用 SCSI バスを装備した OpenVMS Galaxy を構築する場合，次のことに注意する必要があります。

OpenVMS では，各インスタンスで SCSI 装置に同じ名前を正しく付けるには，OpenVMS の装置命名機能を使用する必要があります。

たとえば，SHOW CONFIG コマンドを入力したときに，システムに次のアダプタがあるとしましょう。

```
PKA0 (embedded SCSI for CDROM)
PKB0 (UltraSCSI controller KZPxxx)
PKC0 (UltraSCSI controller)
```

このシステムを 2 インスタンス Galaxy に設定すると，ハードウェアは次のようになります。

```
Instance 0
PKA0  (UltraSCSI controller)

Instance 1
PKA0  (embedded SCSI for CDROM)
PKB0  (UltraSCSI controller)
```

共用 SCSI は，インスタンス 0 の PKA0 からインスタンス 1 の PKB0 に接続されます。

LP_COUNT 環境変数を 0 に設定してシステムを初期化すると，SYSGEN パラメータ STARTUP_P1 が MINIMUM に設定されていない限り，システムで OpenVMS をブートすることができません。

これは，LP_COUNT 変数が 0 に設定されている場合，PKB が PKC に接続されるため， 複数のパーティションで初期化するために設定した SCSI 装置名が， LP_COUNT 変数を 0 に設定して初期化するときに正しくなくなるからです。

ブート時に発生する装置構成で，OpenVMS は PKA0 と PKB0 が接続されていることに気付きます。OpenVMS は各装置に同じ割り当てクラスおよび名前が与えられているものと解釈しますが，この場合は同じではありません。

2 インスタンス Galaxy 用に設定された装置名は， コントローラのコンソール名が変更されたために正しく機能しません。

2.12 OpenVMS Galaxy コンピューティング環境でのセキュリティに関する留意事項

shared-everything クラスタ環境では， すべてのセキュリティ・データベース・ファイルがすべてのインスタンス間で共用されます。 このようなクラスタ環境で動作する OpenVMS Galaxy インスタンスは， Galaxy 関連のすべてのセキュリティ・プロファイルに関して， 自動的に一貫性のあるビューを提供します。

すべての Galaxy インスタンスですべてのセキュリティ・データベース・ファイルを共用しないことを選択した場合は，一貫性のあるセキュリティ・プロファイルを実現するには，手動の操作が必要になります。オブジェクトのセキュリティ・プロファイルを変更すると，このオブジェクトにアクセスできるすべてのインスタンスで同様の変更を行わなければなりません。

手動で次々に変更が必要になるため，このような構成は US C2 評価や他の機関による同様の評価の対象になっていません。オペレーティング・システムに対するセキュリティ評価が必要な組織では，1 つの OpenVMS Galaxy 内のすべてのインスタンスが同じクラスタに所属するように設定する必要があります。

2.13 OpenVMS Galaxy インスタンスのタイム・ゾーンでの構成

OpenVMS Galaxy インスタンスは，同一のクラスタ内に存在するとき以外は， 同一のタイム・ゾーン内に置く必要はありません。たとえば， 3 インスタンスの Galaxy 構成における各インスタンスは， 異なるタイム・ゾーンに置くことができます。

2.14 OpenVMS Galaxy プログラムの開発

この後の項では，OpenVMS Galaxy で開発するのに役立つ OpenVMS プログラミング・インタフェースについて説明します。 概念の多くは，従来のシングル・インスタンス OpenVMS システムを拡張したものです。

各章で説明するサービスの C 関数プロトタイプを表示するには， 次のコマンドを入力してください。

```
$ LIBRARY/EXTRACT=STARLET SYS$LIBRARY:SYS$STARLET_C.TLB/OUTPUT=FILENAME
```

その後，表示するサービスの出力ファイルを検索します。

2.14.1 ロック・プログラミング・インタフェース

Galaxy プラットフォームの主な機能の 1 つに，オペレーティング・システムの 複数のインスタンス間でリソースを共用できる機能があります。 この章で説明するサービスは，Galaxy 内の共用リソースへのアクセスの同期をとるために，協調動作スキームを作成する基礎となるものを提供します。

galaxy ***ロック*** は，スピンロックとミューテクスの組み合わせです。 所有されている galaxy ロックを取得しようとする間，スレッドは短期間スピンします。スピン中にロックが使用可能にならないと，スレッドは待ち状態になります。 これは SMP スピンロックと異なります。 SMP スピンロックでは，スピンが時間切れになると，システムがクラッシュしますが， このような動作は Galaxy では認められません。

galaxy ロックの性質上，これらのロックは共用メモリに存在します。 その共用メモリはユーザまたは galaxy ロック・サービスによって割り当てることができます。 ユーザがメモリを割り当てる場合は，ロック・サービスはロックの場所だけを追跡します。ロック・サービスがメモリを割り当てる場合は， ユーザの代わりにメモリを管理します。

execlets の MON バージョンの一部でしかない他のモニタリング・コードと異なり， galaxy ロックのモニタリング・コードは常にロードされます。

galaxy ロックを取り扱うために複数のルーチンが提供されています。 これらのルーチンは，ロックに関係する基礎的な機能だけを提供します。 SMP をサポートするために使用されるスピン・ロックより少し豊富な機能を備えていますが，ロック・マネージャが提供する機能より，はるかに劣ります。 表 2-1 「ロック・プログラミングのための Galaxy システム・サービス」はロック・プログラミングのための OpenVMS Galaxy システム・サービスについてまとめています。 表 2-1「ロック・プログラミングのための Galaxy システム・サービス」と表 2-2「イベント・プログラミングのための Galaxy システム・サービス」のサービスに対する構文は『OpenVMS System Services Reference Manual』で説明しています。

表 2-1 ロック・プログラミングのための Galaxy システム・サービス

システム・サービス	説明
$ACQUIRE_GALAXY_LOCK	OpenVMS Galaxy のロックの所有権を取得します。
$CREATE_GALAXY_LOCK	OpenVMS Galaxy ロック・ブロックを $CREATE_GALAXY_LOCK_TABLE サービスで作成されたロック・テーブルから割り当てます。
$CREATE_GALAXY_LOCK_TABLE	OpenVMS Galaxy ロック・テーブルを割り当てます。
$DELETE_GALAXY_LOCK	OpenVMS Galaxy ロックを無効にし，削除します。
$DELETE_GALAXY_LOCK_TABLE	OpenVMS Galaxy ロック・テーブルを削除します。
$GET_GALAXY_LOCK_INFO	指定されたロックから特定のフィールドを返します。
$GET_GALAXY_LOCK_SIZE	OpenVMS Galaxy ロックの最小サイズと最大サイズを返します。
$RELEASE_GALAXY_LOCK	OpenVMS Galaxy ロックの所有権を解除します。

2.14.2 システム・イベント・プログラミング・インタフェース

特定のシステム・イベントが発生したときに，アプリケーションにそのことを通知するように登録することができます。 たとえば，インスタンスが galaxy に参加したり，CPU が構成セットに参加したときに，そのことを通知できます (構成セットとは，現在のインスタンスに所有され制御されるプロセッサの集まりです)。 イベントが登録されると，アプリケーションは登録されたイベントの発生時にどのように応答するかを判断できます。

表 2-2「イベント・プログラミングのための Galaxy システム・サービス」はイベント・プログラミングのための OpenVMS システム・サービスについてまとめています。

表 2-2 イベント・プログラミングのための Galaxy システム・サービス

システム・サービス	説明
$CLEAR_SYSTEM_EVENT	SYS$SET_SYSTEM_EVENT を呼び出すことで設定された 1 つ以上の通知要求を削除します。
$SET_SYSTEM_EVENT	OpenVMS システム・イベントが発生したときに，それを通知するための要求を設定します。

2.14.3 OpenVMS Galaxy での System Dump Analyzer (SDA) の使用

この節では， OpenVMS Galaxy 環境に特定した System Dump Analyzer (SDA) について説明します。

SDA の使い方の詳細については，『OpenVMS Alpha System Analysis Tools Manual』を参照してください。

2.14.3.1 共用メモリのダンプ

Galaxy インスタンスでシステム・クラッシュが発生した場合， OpenVMS はデフォルトの動作として，障害が発生したインスタンスのプライベート・メモリの内容と，共用メモリの内容をダンプします。 完全なダンプの場合， 共用メモリとプライベート・メモリのすべてのページがダンプされます。 選択型ダンプの場合は， システム・クラッシュが発生した時点で使用されていたページだけがダンプされます。

共用メモリのダンプは，動的 SYSGEN パラメータ DUMPSTYLE のビット 4 をセットすることで無効に設定できます。このビットは，弊社のサポート要員から助言があった場合にだけセットするようにしてください。 このビットをセットすると， システム・クラッシュの原因を判断するのに必要なデータがシステム・ダンプに含まれなくなる可能性があります。

表 2-3「DUMPSTYLE のビットの定義」は，DUMPSTYLE のすべてのビットの定義と， OpenVMS Alpha での各ビットの意味を示しています。 ビットはどの組み合わせでも指定できます。

表 2-3 DUMPSTYLE のビットの定義

ビット	値	説明
0	1	0= 完全なダンプ。物理メモリ全体の内容がダンプ・ファイルに書き込まれる。 1= 選択型ダンプ。ダンプ・ファイルの有用性をできるだけ高め， その一方でディスク空間を節約できるように，メモリの内容が選択的にダンプ・ファイルに書き込まれる (使用中のページだけが書き込まれる)。
1	2	0= 最小コンソール出力。この出力には，バグチェック・コード， クラッシュが発生した CPU，プロセス，イメージの ID，システム日時， ダンプの出力の進行状況を示す一連のドットが含まれる。 1= 完全なコンソール出力。前に説明した最小出力の他に， スタックとレジスタの内容，システム・レイアウトも出力され， ダンプの進行状況を示すためにダンプ中のプロセスの名前なども出力される。
2	4	0= システム・ディスクへのダンプ。 ダンプは SYS$SYSDEVICE:[SYSn.SYSEXE]SYSDUMP.DMP に書き込まれる。 このファイルがない場合は， SYS$SYSDEVICE:[SYSn.SYSEXE]PAGEFILE.SYS に書き込まれる。 1= 別のディスクへのダンプ。ダンプは dump_dev:[SYSn.SYSEXE]SYSDUMP.DMP に 書き込まれる。ただし，DUMP_DEV はコンソール環境変数 dump_dev の値である。
3	8	0= 非圧縮ダンプ。ページはダンプ・ファイルに直接書き込まれる。 1= 圧縮ダンプ。各ページは書き込む前に圧縮されるので，ダンプを書き込むのに 必要な空間と時間を節約できる。一方，ダンプにアクセスするのに必要な時間は 少し長くなる。
4	16	0= 共用メモリをダンプする。 1= 共用メモリをダンプしない。

DUMPSTYLE のデフォルト設定は 0 です (つまり，共用メモリも含めて，完全なダンプを圧縮せずにシステム・ディスクに書き込みます)。 DUMPSTYLE の値が MODPARAMS.DAT に指定されていない場合は，AUTOGEN.COM はシステムのメモリが 128 MB 未満の場合は 1 に (共用メモリも含めて，選択型ダンプを圧縮せずにシステム・ディスクに書き込みます)，128 MB 以上の場合は 9 に (共用メモリも含めて，選択型ダンプを圧縮してシステム・ディスクに書き込みます) 設定します。

2.14.3.2 SDA コマンド・インタフェースの変更または追加についてのまとめ

表 2-4「SDA コマンドの拡張」は，共用メモリおよび OpenVMS Galaxy データ構造を表示するために使う System Dump Analyzer の拡張についてまとめています。 詳細については適切なコマンドを参照してください。

表 2-4 SDA コマンドの拡張

コマンド	説明
SHOW SHM_CPP	デフォルトでは，すべての SHM_CPP の簡略な情報が表示される。
VALIDATE SHM_CPP	デフォルトではすべての SHM_CPP と，接続されている PFN の数と範囲が確認されるが，各 PFN に対してデータベースの内容は確認されない。
SHOW SHM_REG	デフォルトでは，すべての SHM_REG に関する簡略な情報が表示される。
SHOW GSD への /GLXSYS と /GLXGRP の追加	Galaxy データ構造が表示される。
SHOW GMDB	GMDB と NODEB ブロックの内容を表示する。 デフォルトでは，GMDB が詳細表示される。
SHOW GALAXY	GMDB とすべてのノード・ブロックを簡略表示する。
SHOW GLOCK	Galaxy ロック構造を表示する。デフォルトでは，ベース GLOCK 構造が表示される。
SHOW GCT	Galaxy 構成ツリーを表示する。 デフォルトは /SUMMARY。
SHOW PAGE_TABLE と SHOW PROCESS/PAGE_TABLE	ページ・テーブルとプロセス・ページ・テーブルを表示する。

第3章 OpenVMS アプリケーションに対する NUMA の影響

NUMA (nonuniform memory access) とは，特定の物理メモリ位置へのアクセス時間が，すべての CPU で同じにならないというシステムの属性のことです。このアーキテクチャの下で高い性能を実現するためには，(100 パーセントとはいかなくても) 一貫して適切なロケーションを確保する必要があります。

オペレーティング・システムは，ハードウェアをリソース・アフィニティ・ドメイン (RAD) のセットとして扱います。RAD は，共通のアクセス特性を持つ物理リソース (CPU，メモリ，および I/O) のソフトウェア上の分類です。AlphaServer GS80/160/320 システムでは，RAD はクォド・ビルディング・ブロック (QBB) に対応し，AlphaServer ES47/ES80/GS1280 システムでは，2 プロセッサの CPU ボードに対応します。CPU は，自分の RAD の中のメモリについては，他の RAD の中のメモリよりも高速にアクセスすることができます。

OpenVMS が 1 つの RAD のリソース上で実行されている場合には，NUMA による影響はなく，ここでの説明は適用されません。可能性と実現性がある場合，できるだけ 1 つの RAD の中で実行することで NUMA の複雑な影響をなくし，高い性能を引き出すことができます。

NUMA 環境における全体的なシステム性能に関して最も多い質問は，“ すべてに対して一様 ” と “ 少数に対して最適化 ” のどちらを選ぶべきかということです。言い換えると，すべてのプロセスがほぼ同じ性能を持つようにしたいか，特定のプロセスに焦点を当てて，それらを可能な限り効率化したいかということです。OpenVMS の 1 つのインスタンスが複数の RAD 上で実行されるときには (マシン全体，ハード・パーティション，または Galaxy インスタンスを問わない)，必ずこの質問に対する答えを用意しなければなりません。その答えに応じて，構成と管理に関するいくつかの決定を行う必要があるからです。

OpenVMS の省略時の NUMA 動作モードは「すべてに対して一様」です。リソースは，長期的に見ると，システム上のすべてのプロセスが，平均してほぼ同等の性能ポテンシャルを持つように割り当てられます。

「すべてに対して一様」のモードを希望しない場合には，より特殊な「少数に対して最適化」または「専用」環境を実現するためのインタフェースを理解する必要があります。特定のリソースにプロセスとデータを割り当てて，それらの性能ポテンシャルを最大限に高めることが可能です。

NUMA 環境に関する理解を深めるために，この章では以下のことを説明します。

- 基本オペレーティング・システムの NUMA 動作
- アプリケーション・リソースに関する注意事項
- API

3.1 OpenVMS の NUMA への対応

OpenVMS のメモリ管理とプロセス・スケジューリングは，AlphaServer GS Series システム・ハードウェア上でより効率的に動作するように強化されています。

以下に示す強化分野は，それぞれシステムに新しい能力を付加しています。個別に見れば，これらは特定のアプリケーション・ニーズの性能ポテンシャルを高めています。全体として見れば，これらは多様なアプリケーション・ミックスに必要な環境を提供していると言うことができます。新たに対処された分野は次のとおりです。

- プロセス・プライベート・ページの割り当て
- 予約済みメモリ・ページの割り当て
- プロセス・スケジューリング
- 読み込み専用システム・スペース・ページの複製
- 非ページング・プールの割り当て
- ページ割り当てを観察するためのツール

CPU は，同じ RAD 内のメモリの方が， 別の RAD 内のメモリよりも 3 倍の速度で参照することができます。 このため，実行されるコードと参照されるメモリを， 可能な限り同じ RAD に入れておくことが重要となります。 優れた性能を実現するためには，一貫して適切な位置を確保することが鍵となります。 性能を評価するにあたって， プログラマは次のような質問を考慮に入れる必要があります。

- 実行しようとしているコードはどこにあるのか？
- アクセスしようとしているデータはどこにあるのか？
- 使用しようとしている I/O 装置はどこにあるのか？

OpenVMS スケジューラとメモリ管理サブシステムは，以下の方法で， 最適な位置を決定します。

- 個々のプロセスに優先 RAD，すなわちホーム RAD を割り当てます。
- 通常，プロセスはホーム RAD 内の CPU 上にスケジューリングします。
- オペレーティング・システムの読み込み専用コードと一部のデータを個々の RAD に複製します。
- グローバル・ページを RAD 間に分散させます。
- 予約済みメモリを RAD 間にストライピングします。

3.1.1 ホーム RAD

OpenVMS オペレーティング・システムは，プロセスの作成中に， 個々のプロセスにホーム RAD を割り当てます。 これには 2 つの大きな意味があります。 第 1 に，ごく少数の例外を除き， プロセスのホーム RAD に含まれる CPU の 1 つがプロセスを実行することになります。 第 2 に，プロセスが必要とするすべてのプロセス・プライベート・ページが， ホーム RAD に含まれるメモリから割り当てられるようになります。 この組み合わせにより，ローカル・メモリ参照が行われる可能性が最大化されます。

ホーム RAD を割り当てるときの OpenVMS のデフォルトの動作は， プロセスを RAD 間に分散させるというものです。

3.1.2 システム・コードの複製

システムのスタートアップ時には， オペレーティング・システム・コードが各 RAD のメモリに複製されます。 これにより，システム内のプロセスは， システム機能が必要になったときにはローカル・メモリにアクセスするようになります。 この複製は，エグゼクティブ・コードとインストール済み常駐イメージ・コードの粒度ヒント領域の両方について行われます。

3.1.3 グローバル・ページの分散

OpenVMS のデフォルトの動作は，RAD 間でグローバル・ページ (グローバル・セクションのページ) を分散させるというものです。 このアプローチは，システム・スタートアップ時に予約済みメモリとして宣言されたグローバル・ページの割り当ての際にも使用されます。

3.2 アプリケーション・リソースに関する注意事項

アプリケーション環境はそれぞれ異なる性質を持っています。 アプリケーションの構造によって， 望ましい目標を達成するための最適なオプションが決まります。 これらの決定要因の例を以下に示します。

- プロセスの数
- 必要なメモリの量
- プロセス間での共用の量
- 特定の基本オペレーティング・システム機能の使用
- ロックの使用とその位置

絶対的な規則はあまりありませんが，以下の項では， 一般に最適な結果をもたらす基本的な概念と例をいくつか示します。 メモリ・アクセスの (RAD 上での) ローカル化は常に目標となり

ますが，これは必ずしも達成可能ではなく，トレードオフを判断しなければならない可能性が高いと考えられます。

3.2.1 プロセスと共用データ

1 つのグローバル・セクションにアクセスするプロセスが数百あるいは数千もある場合には，オペレーティング・システムのデフォルトの動作を使用するのが適しています。グローバル・セクションのページは，すべての RAD のメモリに均等に分散され，プロセスのホーム RAD の割り当ては CPU 間で均等に分散されます。これは，グローバル・セクションへのアクセスがランダムだった場合に，すべてのプロセスが似たような性能ポテンシャルを持つことになる分散効果，つまり「一様」効果です。どのプロセスも最適化はされませんが，他のプロセスと比べて極端に不利なプロセスも出現しません。

一方，少数のプロセスがグローバル・セクションにアクセスする場合には，4 つの CPU で処理負荷を扱うことができ，1 つの RAD がグローバル・セクション全体を扱える十分なメモリを持っているという前提で，それらのプロセスを 1 つの RAD に配置することができます。これにより，大部分のメモリ・アクセスがローカル化され，配置されたプロセスの性能が強化されます。この戦略は，1 つのプロセスとデータのセットを 1 つの RAD に配置し，別のプロセスとデータのセットを別の RAD に配置するというように，同じシステム上で繰り返し採用することができます。

3.2.2 メモリ

AlphaServer GS シリーズのシステムには，大量のメモリを搭載することができます。可能な限り，大きなメモリ容量を有効に活用してください。たとえば，コードやデータを複数の RAD に複製するようにします。ある程度の分析が必要となり，スペースの無駄に見えるかもしれず，調整も必要となります。しかし，これによって最終的にローカルなメモリ参照が大幅に増えるのであれば，その価値はあります。

RAM ディスク・プロダクトの使用を検討してください。NUMA が使用されている場合でも，メモリ内参照は実際の装置 I/O よりも高性能です。

3.2.3 共用と同期処理

一般に，データを共用するためには同期処理が必要となります。1 つのメモリ位置を調整メカニズムとして使用している場合 (ラッチ，ロック，セマフォなどと呼ばれます) には，それが多くのリモート・アクセスの原因となっていて，競合レベルが高くなると性能を低下させる可能性があります。この種のロックをデータ全体に複数のレベルで分散させることで，リモート・アクセスの量を減らせることがあります。

3.2.4 OpenVMS 機能の使用

一部の基本オペレーティング・システム機能を頻繁に使用すると，これらの機能をサポートするデータが RAD0 のメモリ内に存在しているために，多数のリモート・アクセスが発生します。

3.3 NUMA リソース・アフィニティ・ドメインのバッチ・ジョブ・サポート

このセクションでは，NUMA 環境でのリソース・アフィニティ・ドメイン (RAD) のサポートにおける OpenVMS バッチ処理サブシステムのアップデートについて説明します。

システム管理者は，NUMA 環境のリソース・アフィニティ・ドメイン (RAD) の特定のサポートにバッチ・キューを割り当てることができ，ユーザはバッチ・ジョブを実行する RAD を指定することができます。

これらの新機能の使用は，バッチ実行キューとバッチ・ジョブに制限されます。

DCL コマンドの説明については，『OpenVMS DCL ディクショナリ』を参照してください。

3.3.1 バッチ・キュー・レベルの RAD サポート

DCL コマンドの INITIALIZE/QUEUE，SET/QUEUE，および START/QUEUE で新しい修飾子 /RAD を使用することができます。システム管理者は，キューに割り当てられたバッチ・ジョブを実行する RAD 番号を指定します。

RAD 値は， 0 から SYI$_RAD_MAX_RADS までの正の整数であることが検証されます。SHOW/QUEUE/FULL コマンドの出力に RAD が表示されるようになり， F$GETQUI レキシカル関数は新しい RAD 項目を受け入れるようになりました。

3.3.1.1 例

ここでは，コマンドのシーケンスとバッチ・キューへの効果を説明します。バッチ・キューの変更を示すために，それぞれの例に SHOW コマンドが含まれています。

- 次の INITIALIZE/QUEUE コマンドは， ノード QUEBIT 上で実行するバッチ・キュー BATCHQ1 を作成または再初期化します。 このキューに割り当てられたジョブはすべて，RAD 0 で実行します。

```
$ INITIALIZE/QUEUE/ON=QUEBIT::/BATCH/RAD=0   BATCHQ1

$ SHOW QUEUE/FULL BATCHQ1
Batch queue BATCHQ1, stopped, QUEBIT::
 /BASE_PRIORITY=4 /JOB_LIMIT=1 /OWNER=[SYSTEM] /PROTECTION=(S:M,O:D,G:R,W:S)
 /RAD=0
```

- 次の START/QUEUE コマンドは，割り当てられたすべてのジョブを QUEBIT の RAD 1 で実行するように BATCHQ1 を変更し， キューが処理のためにジョブを受け入れるようにします。

```
$ START/QUEUE/RAD=1 BATCHQ1

$ SHOW QUEUE/FULL BATCHQ1
Batch queue BATCHQ1, idle, on QUEBIT::
  /BASE_PRIORITY=4 /JOB_LIMIT=3 /OWNER=[SYSTEM] /PROTECTION=(S:M,O:D,G:R,W:S)
  /RAD=1
```

- 次の SET/QUEUE コマンドは，割り当てられたすべてのジョブを QUEBIT の RAD 0 で実行するようにバッチ・キューを変更します。 キューに割り当てられた新しいジョブは，RAD 0 で実行します。 キューですでに実行中のジョブは，以前の RAD 値で完了まで実行を続けます。

```
$ SET/QUEUE/RAD=0 BATCHQ1

$ SHOW QUEUE/FULL BATCHQ1
Batch queue BATCHQ1, idle, on QUEBIT::
  /BASE_PRIORITY=4 /JOB_LIMIT=3 /OWNER=[SYSTEM] /PROTECTION=(S:M,O:D,G:R,W:S)
  /RAD=0
```

- バッチ・キューの RAD 値を消去するには，SET/QUEUE/NORAD コマンドを使用します。

```
$ SET/QUEUE/NORAD BATCHQ1

$ SHOW QUEUE/FULL BATCHQ1
Batch queue BATCHQ1, idle, on QUEBIT::
  /BASE_PRIORITY=4 /JOB_LIMIT=3 /OWNER=[SYSTEM] /PROTECTION=(S:M,O:D,G:R,W:S)
```

- RAD の値を返すには，F$GETQUI レキシカル関数を使用します。 -1 の値は，キューに RAD 値が割り当てられていないことを示します。

```
$ WRITE SYS$OUTPUT F$GETQUI("DISPLAY_QUEUE","RAD","BATCHQ1")
 -1
```

3.3.2 ジョブ・レベルの RAD サポート

新しい修飾子 /RAD は，DCL コマンドの SUBMIT と SET/ENTRY に追加されます。

ユーザは登録したバッチ・ジョブを実行する RAD 番号を修飾子の値で指定します。 SHOW ENTRY および SHOW QUEUE/FULL コマンドが強化されて， バッチ・ジョブの RAD 設定をリストするようになりました。

3.3.2.1 例

RAD 設定のないバッチ・キューにジョブが登録されたときには， そのジョブは SUBMIT コマンドで指定された RAD を使用して実行します。

次のコマンドは，TEST.COM をキュー ANYRADQ に登録します。 ANYRADQ キューには RAD 設定はありません。

```
$ SUBMIT/HOLD/QUEUE=ANYRADQ /RAD=1  TEST.COM
Job TEST (queue ANYRADQ, entry 23) holding

$ SHOW ENTRY/FULL 23
  Entry  Jobname         Username     Blocks  Status
  -----  -------         --------     ------  ------
     23  TEST            SYSTEM               Holding
         On idle batch queue ANYRADQ
         Submitted 24-JUL-2001 14:19:37.44 /KEEP /NOPRINT /PRIORITY=100 /RAD=0
         File: _$1$DKB200:[SWEENEY.CLIUTL]TEST.COM;1
```

RAD 設定のあるバッチ・キューにジョブが登録されたときには，そのジョブは，SUBMIT コマンドで指定された RAD に関係なく， キューに対して指定された RAD を使用して実行します。この動作は，他のバッチ・システム機能と一貫性があります。

キュー BATCHQ1 は，/RAD=0 として定義されています。 次の SUBMIT コマンドの例では，ユーザは登録時に RAD 1 を指定しましたが， RAD 0 で実行するジョブが作成されます。

```
$ SUBMIT/HOLD/QUEUE=BATCHQ1 /RAD=1  TEST.COM
Job TEST (queue BATCHQ1, entry 24) holding

$ SHOW ENTRY 24/FULL
  Entry  Jobname         Username     Blocks  Status
  -----  -------         --------     ------  ------
     24  TEST            SYSTEM               Holding
         On idle batch queue BATCHQ1
         Submitted 24-JUL-2001 14:23:10.37 /KEEP /NOPRINT /PRIORITY=100 /RAD=0
         File: _$1$DKB200:[SWEENEY.CLIUTL]TEST.COM;2
```

3.3.3 実行時の動作

バッチ・ジョブの RAD を指定すると，ジョブ・コントローラは新しい HOME_RAD 引数がジョブの RAD 値に設定されたプロセスを作成します。

ジョブに対して指定された RAD がターゲット・システムで無効であった場合， ジョブ・コントローラは BADRAD メッセージをオペレータ・コンソールに出力します。 正しくない RAD 値がバッチ・キューの RAD 設定に一致した場合，バッチ・キューは停止します。ジョブはキューに残ります。

3.3.3.1 エラー処理

次の例は，実行時処理のエラーを示しています。

```
SYSTEM@QUEBIT> SUBMIT/NONOTIFY/NOLOG/QUEUE=BATCHQ1 TEST.COM
Job TEST (queue BATCHQ1, entry 30) started on BATCHQ1

OPCOM MESSAGES

SYSTEM@QUEBIT> START/QUEUE BATCHQ1
%%%%%%%%%%%%  OPCOM  25-JUL-2001 16:15:48.52  %%%%%%%%%%%%
Message from user SYSTEM on QUEBIT
```

```
%JBC-E-FAILCREPRC, job controller could not create a process

%%%%%%%%%%  OPCOM  25-JUL-2001 16:15:48.53  %%%%%%%%%%
Message from user SYSTEM on QUEBIT
-SYSTEM-E-BADRAD, bad RAD specified

%%%%%%%%%%  OPCOM  25-JUL-2001 16:15:48.54  %%%%%%%%%%
Message from user SYSTEM on QUEBIT
%QMAN-E-CREPRCSTOP, failed to create a batch process, queue BATCHQ1 will be stopped

$ SYSTEM@QUEBIT> WRITE SYS$OUTPUT  -
_$ F$message(%x'F$GETQUI("DISPLAY_ENTRY","CONDITION_VECTOR","30")')
%SYSTEM-E-BADRAD, bad RAD specified
```

3.3.3.2 バッチ・キューの RAD の変更

バッチ・キューの RAD 値を変更しても，バッチ・キューに登録されているジョブは， キューに対して指定された新しい RAD 値で動的に更新されるわけではありません。

実行中のジョブは，元の RAD 値を使用して処理を続行します。 保留中または時間切れ実行状態のジョブは，元の RAD 値のままです。ただし，そのようなジョブが実行可能になると，ジョブは新しい RAD 値で更新され， キューに対して指定された RAD で実行します。

3.4 RAD アプリケーション・プログラミング・インタフェース

アプリケーション・プログラマとシステム管理者は，システムの省略時の値が運用環境のニーズを満たせない場合には， RAD に固有の多数のインタフェースを使ってプロセスとメモリのロケーションを制御することができます。 以下の項は簡単な説明です。 詳細については『OpenVMS System Services Reference Manual』を参照してください。

プロセスの作成

プロセスに特定のホーム RAD を持たせたい場合は， SYS$CREPRC システム・サービスの新しい HOME_RAD 引数を使用します。これにより，アプリケーションからロケーションを制御することができます。

プロセスの移動

プロセスがすでに作成されており，これを再配置したい場合には，システム・サービス SYS$PROCESS_AFFINITY または SYS$PROCESS_CAPABILITY で CAP$M_PURGE_WS_IF_NEW_RAD フラグを使用します。アフィニティまたはケーパビリティの選択によってプロセスのホーム RAD が変わる場合には，プロセスのワーキング・セットは削除されます。

プロセスに関する情報の取得

SYS$GETJPI システム・サービスは，プロセスのホーム RAD を返します。

グローバル・セクションの作成

SYS$CRMPSC_GDZRO_64 および SYS$CREATE_GDZRO システム・サービスは RAD 引数マスクを受け付けます。 これは，OpenVMS がグローバル・セクションのページをどの RAD に割り当てるかを指定します。

予約済みメモリの割り当て

予約済みメモリの割り当てのための SYSMAN インタフェースは RAD 修飾子を持っているので，システム管理者は特定の RAD のメモリを予約することができます。

システムに関する情報の取得

SYS$GETSYI システム・サービスは， RAD 情報を獲得するために使われる以下の項目コードを定義しています。

- RAD_MAX_RADS は，プラットフォーム上で使用可能な RAD の数の最大値を示します。
- RAD_CPUS は RAD/CPU のペアのロングワード配列を示します。
- RAD_MEMSIZE は RAD/page_count のペアのロングワード配列を示します。
- RAD_SHMEMSIZE は RAD/page_count のペアのロングワード配列を示します。

RAD_SUPPORT システム・パラメータ

RAD_SUPPORT システム・パラメータは，個々の RAD 関連の動作をカスタマイズするために定義されている多くのビットとフィールドを持っています。これらのビットについての詳細は，3.7.1 項「SHOW RAD」の例を参照してください。

3.5 RAD システム・サービスの要約の表

次の表は， OpenVMS Version 7.3 以降の RAD システム・サービスに関する情報を示しています。

詳細については，『OpenVMS System Services Reference Manual』を参照してください。

システム・サービス	RAD 情報
$CREATE_GDZRO	引数: **rad_mask** フラグ: **SEC$M_RAD_HINT** エラー状態: **SS$_BADRAD**
$CREPRC	引数: **home_rad** 状態フラグ・ビット: **stsflg** シンボリック名: **PRC$M_HOME_RAD** エラー状態: **SS$_BADRAD**
$CRMPSC_GDZRO_64	引数: **rad_mask** フラグ: **SEC$M_RAD_MASK** エラー状態: **SS$_BADRAD**
$GETJPI	項目コード: **JPI$_HOME_RAD**
$GETSYI	項目コード: **RAD_MAX_RADS**，**RAD_CPUS**，**RAD_MEMSIZE**，**RAD_SHMEMSIZE**，**GALAXY_SHMEMSIZE**
$SET_PROCESS_PROPERTIESW	項目コード: **PPROP$C_HOME_RAD**

3.6 RAD DCL コマンドの要約の表

次の表は，OpenVMS RAD DCL コマンドを要約したものです。 詳細については，『OpenVMS DCL ディクショナリ』を参照してください。

DCL コマンド/レキシカル	RAD 情報
SET PROCESS	修飾子: **/RAD=HOME=n**
SHOW PROCESS	修飾子: **/RAD**
F$GETJPI	項目コード: **HOME_RAD**
F$GETSYI	項目コード: **RAD_MAX_RADS**，**RAD_CPUS**，**RAD_MEMSIZE**，**RAD_SHMEMSIZE**

3.7 System Dump Analyzer (SDA) の RAD サポート

以下に示す System Dump Analyzer (SDA) コマンドには RAD サポートが追加されています。

- SHOW RAD
- SHOW RMD (予約済みメモリ記述子)
- SHOW PFN

3.7.1 SHOW RAD

SDA コマンドの SHOW RAD は，以下の情報を表示します。

- RAD_SUPPORT システム・パラメータ・フィールドの設定と説明
- CPU とメモリの RAD への割り当て

このコマンドは，RAD をサポートするハードウェア・プラットフォームでのみ有効です (AlphaServer GS160 システムなど)。デフォルトの設定では，SHOW RAD コマンドは RAD_SUPPORT システム・パラメータ・フィールドの設定を表示します。

形式:

```
SHOW RAD [number|/ALL]
```

パラメータ:

number

指定された RAD の CPU とメモリに関する情報を表示します。

修飾子:

/ALL

RAD_SUPPORT パラメータ・フィールドの設定とすべての CPU/ メモリ割り当てを表示します。

次の例は，RAD_SUPPORT システム・パラメータ・フィールドの設定を示しています。

```
SDA> SHOW RAD

Resource Affinity Domains
-------------------------

RAD information header address: FFFFFFFF.82C2F940
Maximum RAD count:                      00000008
RAD containing SYS$BASE_IMAGE:          00000000
RAD support flags:                      0000000F

3         2 2         1 1
1         4 3         6 5         8 7         0
+-----------+-----------+-----------+-----------+
|..|..| skip|ss|gg|ww|pp|..|..|..|..|..|fs|cr|ae|
+-----------+-----------+-----------+-----------+
|..|..|    0| 0| 0| 0| 0|..|..|..|..|..|00|11|11|
+-----------+-----------+-----------+-----------+

Bit 0 = 1:          RAD support is enabled

Bit 1 = 1:          Soft RAD affinity support is enabled
                    (Default scheduler skip count of 16 attempts)

Bit 2 = 1:          System-space replication support is enabled

Bit 3 = 1:          Copy on soft fault is enabled

Bit 4 = 0:          Default RAD-based page allocation in use

                    Allocation Type               RAD choice
                    ---------------               ----------
                    Process-private pagefault     Home
                    Process creation or inswap    Random
                    Global pagefault              Random
                    System-space page allocation  Current

Bit 5 = 0:          RAD debug feature is disabled)
```

次の例は，RAD 2 の CPU とメモリに関する情報を示しています。

```
SDA> SHOW RAD 2
```

```
Resource Affinity Domain 0002
-----------------------------

CPU sets:

  Active      08 09 10 11
  Configure   08 09 10 11
  Potential   08 09 10 11


PFN ranges:

  Start PFN   End PFN     PFN count   Flags
  ---------   --------    ---------   -----
  01000000    0101FFFF    00020000    000A  OpenVMS Base
  01020000    0103FFFF    00020000    0010  Galaxy_Shared

SYSPTBR:      01003C00)
```

3.7.2 SHOW RMD (予約済みメモリ記述子)

SDA コマンドの SHOW RMD は，予約済みメモリの割り当て元の RAD を示すように 拡張されています。予約済みメモリの割り当て時に RAD が指定されていなかった場合， SDA は ANY と表示します。

3.7.3 SHOW PFN

SDA コマンドの SHOW PFN には /RAD 修飾子が追加されています。 これは既存の /COLOR 修飾子に似ています。

3.7.4 RAD のハード・アフィニティのサポート

SET PROCESS コマンドが強化されて，/AFFINITY 修飾子が追加されました。 /AFFINITY 修飾子によって，カーネル・スレッドのアフィニティ・マスクのビットをセットまたはクリアできます。 アフィニティ・マスクのビットは，個別に操作することも， グループ単位や全部を一括して操作することもできます。

/NOAFFINITY 修飾子は，/PERMANENT 修飾子の設定に基づいて， 現在またはパーマネント・アフィニティ・マスクでセットされているすべてのアフィニティ・ビットをクリアします。 /AFFINITY 修飾子を指定しても直接の効果はなく， 次の 2 次パラメータによって指定された操作のターゲットを示すだけです。

- /SET=(n[,...])

  CPU ID n によって指定された現在アクティブな CPU のアフィニティをセットします。n の範囲は，0 から 31 までです。

- /CLEAR=(n[,...])

  位置の値 n によって指定された現在アクティブな CPU のアフィニティをクリアします。n の範囲は，0 から 31 までです。

- /PERMANENT

  現在のアフィニティ・マスクだけでなく，パーマネント・アフィニティ・マスクに対しても操作を実行して， カーネル・スレッドの終了まで，変更を有効にします (省略時の動作では，実行中のイメージのアフィニティ・マスクだけが影響を受けます)。

次の例は， CPU 1，2，3，および 5 のアフィニティ・ビットをアクティブにセットする方法を示しています。

```
$ SET PROCESS /AFFINITY /PERMANENT /SET = (1,2,3,5)
```

RAD をサポートしているシステムでは，1 つのプロセスに対するアフィニティを設定する CPU のセットは同じ RAD 内に存在していなければなりません。 たとえば，RAD 0 に CPU 0，1，2，3 があり，RAD 1 に CPU 4，5，6，7 がある AlphaServer GS160 では，SET = 2,3,4,5 は良い設定では**ありません**。全体の半分が，ホーム RAD の外で実行されることになるからです。

第4章 AlphaServer GS140/GS60/GS60E システムでの OpenVMS Galaxy の構築

OpenVMS Alpha バージョン 7.3 では，AlphaServer GS60， GS60E システムおよび GS140 システムでは OpenVMS Galaxy 構成がサポートされます。 AlphaServer GS140 システムでは OpenVMS の 3 つのインステンスを実行でき， AlphaServer GS60/GS60E システムでは 2 つのインスタンスが実行できます。

4.1 ファームウェアのダウンロード

AlphaServer GS60， GS60E および GS140 システムで OpenVMS Galaxy 環境を構築するには， 次の場所から V6.2 コンソール・ファームウェアの最新のバージョンをダウンロードする必要があります。

http://ftp.digital.com/pub/DEC/Alpha/firmware/

4.2 OpenVMS Galaxy の構築

このファームウェアがあると，次の操作が可能です。

- 第5章 「AlphaServer 8400 システムでの OpenVMS Galaxy の構築」に説明されている手順に従って，AlphaServer GS140 で OpenVMS Galaxy コンピューティング環境を構築できます。
- 第6章 「AlphaServer 8200 システムでの OpenVMS Galaxy の構築」に説明されている手順に従って，AlphaServer GS60 または GS60E システムで OpenVMS Galaxy コンピューティング環境を構築できます。

第5章 AlphaServer 8400 システムでの OpenVMS Galaxy の構築

この章では，AlphaServer 8400 で OpenVMS Galaxy コンピューティング環境を 構築するプロセスについて説明します。

5.1 ステップ 1: 構成の選択とハードウェア要件の判断

AlphaServer 8400 Galaxy 構成の概要

次の目的で使用される 9 つのスロット:

- I/O モジュール (KFTIA タイプまたは KFTHA タイプ)
- メモリ・モジュール
- プロセッサ・モジュール (モジュールごとに 2 つの CPU)

各パーティションのコンソール・ライン:

- 最初のパーティション用の標準 UART
- 各追加パーティション用の KFE72-DA

ルール:

- 各パーティションに I/O モジュールが 1 つ必要です。
- I/O モジュールは最大 3 つまで認められます。
- 各パーティションに少なくとも 1 つの CPU モジュールが必要です。

構成例 1

2 つのパーティション，8 つの CPU，12 GB のメモリ

- 9 つのスロットは次のように割り当てられます。
  - 2 つの I/O モジュール
  - 4 つのプロセッサ・モジュール (それぞれ 2 つの CPU)
  - 3 つのメモリ・モジュール (各 4GB)

構成例 2

3 つのパーティション，8 つの CPU，8 GB のメモリ

- 9 つのスロットは次のように割り当てられます。
  - 3 つの I/O モジュール
  - 4 つのプロセッサ・モジュール (それぞれ 2 つの CPU)
  - 2 つのメモリ・モジュール (各 4GB)

5.2 ステップ 2: ハードウェアの設定

構成に必要なハードウェアを入手した後， この節の手順に従ってハードウェアを組み立てます。

5.2.1 KFE72-DA コンソール・サブシステム・ハードウェアの概要

AlphaServer 8400 では，標準のビルト・イン UART が提供されます。 これはプライマリ Galaxy インスタンスのコンソール・ラインとして使用されます。 追加の各インスタンスのコンソールでは， KFE72-DA コンソール・サブシステムが必要であり， これは追加コンソール・ポートを設定する EISA-bus モジュールの集合です。

AlphaServer 8400 は最大 3 つの I/O モジュールをサポートします。 3 つより多くのモジュールを構成することはできません。

各 KFE72-DA サブシステムは別々の DWLPB カード・ケージに取り付け， それを接続するホースを KFTIA または KFTHA タイプの別の I/O モジュールに取り付けます。

最初にスロット 8 から順に，すべての KFTIA I/O モジュールを取り付ける必要があります。KFTHA I/O モジュールは KFTIA モジュールの後に取り付けなければならず，番号の小さいスロットから順に取り付けます。

このスロット割り当てルールに従っていれば， これらの 2 つの I/O モジュールを任意の組み合わせで使用できます。

コンソール・サブシステムを構成する場合は， I/O モジュールと DWLPB カード・ケージを接続する I/O ホースを一番下のホース・ポートに接続しなければなりません。 単に一番下の **使用可能な** ホース・ポートではなく， 絶対的な場所が最初のホース・ポートです。 つまり，モジュールの一番上に最も近いポートです。

KFE72-DA には 3 つの EISA モジュールがあり，次のポートを提供します。

- 2 つの COM ポート
- イーサネット・ポート
- 小型スピーカーと他のポート (キーボードやマウスなど，Galaxy 構成に対しては使用されません)

5.2.2 KFE72-DA モジュールの取り付け

インスタンス 0 の後の OpenVMS オペレーティング・システムの各インスタンスに対して，PCI カード・ケージに次の 3 つのモジュールを取り付けなければなりません。

- 標準 I/O モジュール
- シリアル・ポート・モジュール
- コネクタ・モジュール

これらのモジュールを取り付けるには， 5.2.2.1 項「PCI カードをケージから取り出す」～ 5.2.2.3 項「コネクタの接続」の手順を行ってください。 これらの手順は，『KFE72 Installation Guide』の第 5 章の KFE72-DA モジュールの取り付け手順を補足するものです。

5.2.2.1 PCI カードをケージから取り出す

『KFE72 Installation Guide』の第 5.2.1 項の手順を行います。

5.2.2.2 モジュールを挿入し，リボン・ケーブルを接続する

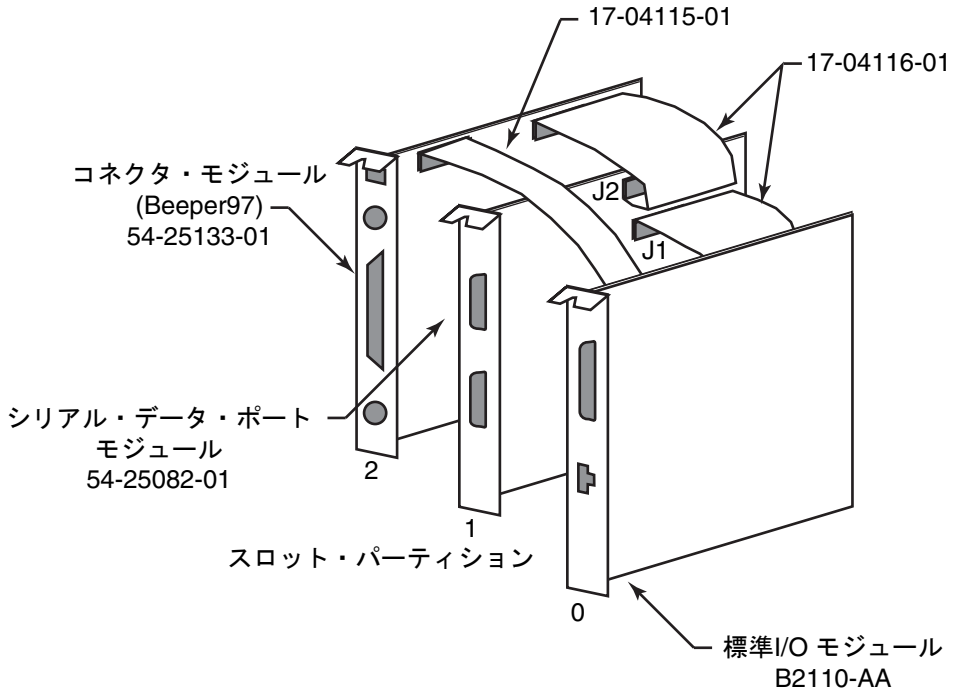
注意:

PCI モジュールを取り付ける場合， オプションのバルクヘッドが PCI カード・ケージの EMI ガスケットと対応することを確認してください。

KFE72-DA モジュールは DWLPB カード・ケージのスロット 0，1，2 に取り付けなければなりません。

モジュールを PCI カード・ケージに挿入し，適切なリボン・ケーブルを接続するには， 図 5-1「リボン・ケーブルの接続」 を参照し，次の操作を行います。

図 5-1 リボン・ケーブルの接続



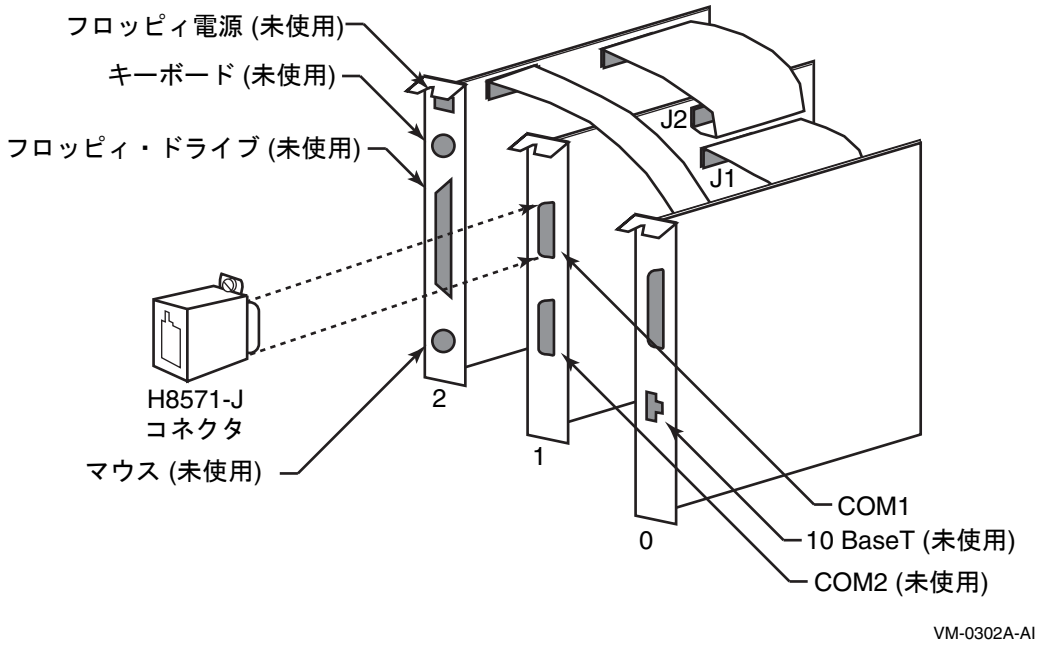
1. 標準 I/O モジュール (B2110-AA) をスロット 0 に挿入します。 モジュールをカード・ケージにネジで固定します。
2. 60 ピン・リボン・ケーブル (17-04116-01) をシリアル・ポート・モジュール (54-25082-01) のコネクタ J1 と J2 に接続します。
3. シリアル・ポート・モジュール (54-25082-01) をスロット 1 に挿入します。 モジュールをカード・ケージにネジで固定します。
4. 60 ピン・リボン・ケーブル (17-04116-01) をコネクタ J1 のシリアル・ポート・モジュールから標準 I/O モジュールに接続します。
5. コネクタ・モジュールをスロット 2 に挿入します。
6. 34 ピン・リボン・ケーブル (17-04115-01) をスロット 0 の標準 I/O モジュール (B2110-AA) とスロット 2 のコネクタ・モジュール (54-25133-01) の間で接続します。
7. 60 ピン・リボン・ケーブル (17-04116-01) をコネクタ J2 のシリアル・ポート・モジュールからコネクタ・モジュールに接続します。

5.2.2.3 コネクタの接続

コンソール・ターミナルと追加装置を接続するには，図 5-2「コネクタ」を参照し，コンソール・シリアル・ライン (H8571-J コネクタ) を COM1 に接続します。

シリアル・ポート・モジュールの番号 1 と番号 2 の間の 2 つの矢印は，シリアル・ポートに対する業界標準記号であり，ポート番号を示しているわけではありません。

図 5-2 コネクタ



5.2.3 シェルフをシステムに取り付ける

カード・ケージを取り付けるには，『KFE72 Installation Guide』の第 5.2.3 項の手順のステップ 2〜9 を実行します。

5.2.4 ターミナル・サーバの使用

ターミナル・サーバを使用してコンソール・ラインをまとめなければならないことがあります。 たとえば，DECserver200 を使用すると，ネットワーク上の各コンソールに逆方向 LAT アクセスすることができます。 この操作がそれほど必要でない場合は，OpenVMS Galaxy 構成の管理を大幅に簡単にできます。 LAT サーバや他のターミナル・コンセントレータの構成の詳細については， 該当する製品のマニュアルを参照してください。

5.2.5 EISA 装置の取り付け

プラグイン EISA 装置はパーティション 0 でのみ構成できます。 EISA 装置を取り付けた後，EISA Configuration Utility (ECU) を実行するように要求するメッセージがコンソールから出力されます。

次のように行って ECU を実行します。

1. すべての OpenVMS Galaxy インスタンスをシャットダウンします。
2. フロッピー・ディスク・ドライブがプライマリ・パーティション・ハードウェアに正しく接続されているかどうか確認します。通常，ドライブは PCI スロット 2 のコネクタ・モジュール (Beeper の部品番号は 54-25133-01) にケーブル接続することができます。
3. ECU イメージを格納したフロッピーを挿入します。
4. プライマリ・コンソールから次のコマンドを入力します。

```
P00>>> SET ARC_ENABLE ON
P00>>> INITIALIZE
P00>>> RUN ECU
```

5. ECU から要求される手順を実行します。
6. プライマリ・コンソールから次のコマンドを入力します。

```
P00>>> boot
$ @SYS$SYSTEM:SHUTDOWN
```

```
P00>>> SET ARC_ENABLE OFF
P00>>> INITIALIZE
P00>>> LPINIT
```

7. OpenVMS Galaxy をリブートします。

ECU には 2 つのバージョンがあり，1 つはグラフィックス端末で実行され， もう 1 つはキャラクタ・セル端末で実行されます。どちらのバージョンもフロッピーに格納されており，コンソールはどちらのバージョンを実行するかを判断します。OpenVMS Galaxy システムの場合，プライマリ・コンソールは常にキャラクタ・セル端末を装備したシリアル装置です。

ECU を実行しないと，OpenVMS は次のメッセージを表示します。

```
      %SYSTEM-I-NOCONFIGDATA, IRQ Configuration data for EISA
slot xxx was not found, please run the ECU and reboot.
```

このメッセージを無視すると，システムはブートされますが， プラグイン EISA 装置は無視されます。

これまでの説明に従って OpenVMS Galaxy ハードウェアを構成し，設定した後， 次の手順を実行して OpenVMS Galaxy インスタンスをインストールし，ブートします。

5.3 ステップ 3: システム・ディスクの作成

インスタンスごとにシステム・ディスクを使用するのか，**または** クラスタ全体で共通のディスクで使用するのかを判断します。

OpenVMS バージョン 7.1-2 より以前のバージョンを稼働しているクラスタ・メンバのうち，Galaxy インスタンスと同じ VMS$OBJECTS.DAT ファイルを共用するクラスタ・メンバの場合は，新しい SECURITY.EXE が必要です。

5.4 ステップ 4: OpenVMS Alpha のインストール

OpenVMS Galaxy ソフトウェアを実行するのに，特別なインストール手順は必要ありません。Galaxy 機能は基本オペレーティング・システムに組み込まれており， この章で説明するコンソール・コマンドとシステム・パラメータ値を使用して， 有効または無効に設定できます。

OpenVMS Alpha オペレーティング・システムのインストールの詳細については，『OpenVMS インストレーション・ガイド [翻訳版]』を参照してください。

5.4.1 OpenVMS Galaxy ライセンス情報

ライセンス管理については，『OpenVMS License Management Utility Manual』を参照してください。

5.5 ステップ 5: ファームウェアのアップグレード

AlphaServer 8400 で OpenVMS Galaxy 環境を構築するには， 各プロセッサ・モジュールでファームウェアのアップグレードが必要です。 これらのモジュールを Galaxy 以外の構成で再び使用する場合は， 以前のファームウェアを再インストールする必要があります。 現在のファームウェア CD は必ず保管しておいてください。

使用する予定の**すべての**プロセッサ・モジュールを取り付け，同時に更新しておけば，作業時間を短縮できます。 AlphaServer 8400 では， すべてのプロセッサ・ボードで同じファームウェアを使用しなければなりません。 後でボードをアップグレードしなければならない場合は，次の操作が必要です。

1. ファームウェア・リビジョン・レベルが異なるすべてのボードを取り外します。
2. 以前のボードを更新します。
3. 残りのボードを再び取り付けます。

ファームウェアをアップグレードするには，システムの電源をオンにし， 非 Galaxy モードで稼働します (つまり，LP_COUNT コンソール環境変数を設定している場合，0 に設定する必要があります)。

コンソール環境変数を設定するには，次のコマンドを入力します。

```
P00>>> SET LP_COUNT 0
P00>>> INIT
```

ファームウェアをアップグレードするには，AlphaSystem Engineering から提供されている標準コンソール・ファームウェア・アップデートを使用します。最新版のファームウェアをダウンロードするには， 下記の Alpha Systems Firmware Web サイトをご覧ください。

http://ftp.digital.com/pub/DEC/Alpha/firmware/

5.6 ステップ 6: 環境変数の設定

すべてのプロセッサ・モジュールでファームウェアをアップグレードした後，次の例に示すように，Galaxy 固有の環境変数を作成できます。 この例では，2 つのインスタンス，8 つの CPU， 1 GB の OpenVMS Galaxy コンピューティング環境を構成しているものと仮定しています。

```
P00>>> create -nv lp_count          2
P00>>> create -nv lp_cpu_mask0      1
P00>>> create -nv lp_cpu_mask1      fe
P00>>> create -nv lp_io_mask0       100
P00>>> create -nv lp_io_mask1       80
P00>>> create -nv lp_mem_size0      10000000
P00>>> create -nv lp_mem_size1      10000000
P00>>> create -nv lp_shared_mem_size  20000000
P00>>> init
```

これらの変数を作成した後，コンソール SET コマンドを使用して， 変数を操作することができます。 これらの変数はプロセッサ 0 でのみ作成する必要があります。

ここでは各環境変数について詳しく説明します。

LP_COUNT number

この変数が 0 に設定されていると，システムは従来の SMP 構成だけをブートします。 Galaxy コンソール・モードは OFF になります。

この変数が 0 以外の値に設定されている場合は，Galaxy 機能が使用され， Galaxy 変数が解釈されます。LP_COUNT の正確な値は， コンソールが認識する Galaxy パーティションの数を表します。 この値は 0，2，3 のいずれかでなければなりません。

3 つのパーティションに対してリソースを割り当て，この変数を 2 に設定すると， 残りのリソースは割り当てられないままになります。割り当てられない CPU はパーティション 0 に割り当てられます。また，あらかじめ最大数のパーティションに対して変数を作成し，必要になるまでリソースを割り当てないようにすることもできます (つまり，変数を 0 以外の値に設定します)。

LP_CPU_MASKn mask

このビット・マスクは，指定された Galaxy パーティション番号にどの CPU を最初に割り当てるかを指定します。 AlphaServer 8400 コンソールは，パーティション内の最初の偶数番号の CPU を初期インスタンスとして CPU 0 で始まるプライマリとして選択します。 リソースを割り当てる場合は，このことに注意してください (つまり，奇数番号だけの CPU をパーティションに割り当てないでください)。

LP_IO_MASKn mask

これらの変数は， スロット番号によって I/O モジュールを各インスタンスに割り当てます。

- 100 はスロット 8 の I/O モジュールを表します。
- 80 はスロット 7 の I/O モジュールを表します。

- 40 はスロット 6 の I/O モジュールを表します。

AlphaServer 8400 の場合，ここに示した割り当てだけが有効です。

これらのマスクを使用して複数の I/O モジュールをインスタンスに割り当てることができますが，各 Galaxy インスタンスは少なくとも 1 つの I/O モジュールを 必要とします。

LP_MEM_SIZEn size

これらの変数は，指定されたインスタンスに対して特定の容量のプライベート・メモリを割り当てます。 システム内のメモリ容量と各インスタンスにとって必要な割り当てをもとに， 適切な値を使用してこれらの変数を作成することが必要です。 一般的な値については，付録 B「メモリ・サイズ設定の共通値」を参照してください。

また，この後の項目の共用メモリ変数も参照してください。

LP_SHARED_MEM_SIZE size

この変数は，共用メモリとして使用するメモリを割り当てます。 一般的な値については，付録 B「メモリ・サイズ設定の共通値」を参照してください。

注意:

共用メモリは 8 MB の倍数で割り当てなければならず， すべての値は 16 進バイトで表現されます。

使用する共用メモリの容量だけを定義でき，他の LP_MEM_SIZE 変数は未定義のままにしておくことができます。 このようにすると，コンソールは上位アドレス空間から共用メモリを割り当て，LP_COUNT 変数によって指定された数のパーティションに対して，残りのメモリを等しく分割します。 また，LP_MEM_SIZE 変数を使用して特定のパーティションにメモリを明示的に割り当て，他のパーティションのメモリ割り当てを未定義のままにした場合も， コンソールはメモリ・フラグメントを，明示的にメモリが割り当てられたパーティションおよび共用メモリに対して割り当て，残りのメモリを分割して， 明示的にメモリが割り当てられていない残りのパーティションに割り当てます。

BOOTDEF_DEV 変数と BOOT_OSFLAGS 変数

初期インストールの後や，システムの障害やオペレータによって要求されたリブートの後，AUTOGEN が正しくリブートされるように， ブートの前に各 Galaxy コンソールでこれらの変数を設定する必要があります。

Galaxy 環境変数の例

```
P00>>> SHOW LP*

lp_count 2
lp_shared_mem_size 20000000   (512 MB)
lp_mem_size0 10000000 (256 MB)
lp_mem_size1 10000000 (256 MB)
lp_cpu_mask0 1 (CPU 0)
lp_cpu_mask1 fe (CPUs 1-7)
lp_io_mask0 100 (I/O module in slot 8)
lp_io_mask1 80 (I/O module in slot 7)

P00>>
```

5.7 ステップ 7: セカンダリ・コンソール装置の起動

KFE72-DA が Windows-NT に構成されていた場合は， ビデオ・ボードを検出しようとしますが，検出できないときはハングします。 OpenVMS Galaxy を構成する場合，このような状況が一般的に発生します。 操作モードを設定するには，コンソール・コマンドを使用します。

```
P00>>> SET CONSOLE SERIAL
```

セカンダリ・コンソールを初期化する前に，プライマリ・コンソールに対してこのコマンドを入力すると， 設定はセカンダリ・コンソール・ハードウェアに伝達されます。

イーサネット・ポートを使用する場合は，使用するメディアの種類と接続をコンソールに通知しなければなりません。つまり，AUI，UDP，ツイスト・ペアのいずれを使用するのかを指定する必要があります。コンソールとオペレーティング・システムはどのメディアを使用するかを判断しますが，次のコマンドを入力すれば，特定のメディア・タイプを割り当てることができます。

```
P00>>> SHOW NETWORK

P00>>> SET EWA0_MODE TWISTED
```

最初のコマンドは，使用可能なネットワーク装置の一覧を表示します。2 番目のコマンドは，指定された装置 (この例では EWA0) に対してデフォルト・メディア・タイプを設定します。これはセカンダリ・コンソールを初期化する前に， すべてのイーサネット装置に対して実行しなければなりません。

コンソール・モードとネットワーク・メディア・タイプ (使用する場合) を設定した後，システムを再初期化して，現在の設定を保存します。 Galaxy パーティションをすでに定義している場合は， ここで初期化することができます。Galaxy パーティションをまだ定義していない場合は， 初期化は後で実行しなければなりません。

システムを初期化できる場合は，次のコマンドを入力します。

```
P00>>> INIT
```

プライマリ・コンソールからの応答として，通常の電源投入時の自己診断テスト (POST) レポートが出力されます。これには最大 2 分かかる可能性があります。Galaxy パーティションを適切に定義した場合は，プライマリ・パーティションに関連する I/O 装置だけが表示されます。

パーティションが定義されていることを確認するには，次のコマンドを入力します。

```
P00>>> SHOW DEVICE
```

または

```
P00>>> SHOW NETWORK
```

セカンダリ・コンソールを初期化するには，次のコマンドを入力します。

```
P00>>> LPINIT
```

コンソールに次の情報が表示されます。

```
Partition 0: Primary CPU = 0
Partition 1: Primary CPU = 2
Partition 0: Memory Base = 000000000   Size = 010000000
Partition 1: Memory Base = 010000000   Size = 010000000
Shared Memory Base = 020000000   Size = 010000000
LP Configuration Tree = 12c000
starting cpu 1 in Partition 1 at address 01000c001
starting cpu 2 in Partition 1 at address 01000c001
starting cpu 3 in Partition 1 at address 01000c001
starting cpu 4 in Partition 1 at address 01000c001
starting cpu 5 in Partition 1 at address 01000c001
starting cpu 6 in Partition 1 at address 01000c001
starting cpu 7 in Partition 1 at address 01000c001

P00>>>
```

このコマンドは **プライマリ** `Galaxy` **コンソール** から入力しなければなりません。 Galaxy パーティションが正しく定義されており，ハードウェア・リソースが正しく構成されている場

合は，プライマリ・コンソールは各セカンダリ・パーティションに割り当てられているプロセッサを起動するはずです。 各セカンダリ・コンソールは 2 分以内に初期化されます。

1 つ以上のコンソールの初期化が失敗した場合は，ハードウェアの取り付け，Galaxy パーティションの定義， ハードウェアの割り当てを二重にチェックする必要があります。

OpenVMS コンソールの制限事項とヒントの詳細については， 第12章 「OpenVMS Galaxy に関するヒントと手法」 を参照してください。

5.8 ステップ 8: OpenVMS Galaxy のブート

Galaxy ファームウェアを正しくインストールし，コンソールを構成した後， 次の方法で初期 Galaxy 環境をブートできます。

各 Galaxy インスタンスに対して，次の操作を実行します。

```
P00>>> B -FL 0,1 DKA100 // or whatever your boot device is.

SYSBOOT> SET GALAXY 1

SYSBOOT> CONTINUE
```

構成はこれで終了です。これで OpenVMS Galaxy が構築されました。

第6章 AlphaServer 8200 システムでの OpenVMS Galaxy の構築

この章では，AlphaServer 8200 で OpenVMS Galaxy コンピューティング環境を構築するプロセスについて説明します。 第5章「AlphaServer 8400 システムでの OpenVMS Galaxy の構築」で説明している AlphaServer 8400 の手順と異なる手順を中心に説明しています。

6.1 ステップ 1: 構成の選択とハードウェア要件の判断

AlphaServer 8200 Galaxy 構成の概要

- 2 つのインスタンスのみ
- 次の目的で使用される 5 つのスロット:
  - 2 つのプロセッサ・モジュール (それぞれ 2 つの CPU)
  - 2 つの I/O モジュール
  - 1 つのメモリ・モジュール

6.2 ステップ 2: Galaxy ハードウェアの設定

構成に必要なハードウェアを入手した後，第5章「AlphaServer 8400 システムでの OpenVMS Galaxy の構築」の 5.2.1 項「KFE72-DA コンソール・サブシステム・ハードウェアの概要」から 5.2.4 項「ターミナル・サーバの使用」までの手順に従い，その後，この節の手順に従ってハードウェアを組み立てます。

6.2.1 EISA 装置の取り付け

プラグイン EISA 装置はパーティション 0 でのみ構成できます。 EISA 装置を取り付けた後，EISA Configuration Utility (ECU) を実行するように要求するメッセージがコンソールから出力されます。

次の説明に従って ECU を実行します。

1. すべての OpenVMS Galaxy インスタンスをシャットダウンします。
2. フロッピィ・ディスク・ドライブがプライマリ・パーティション・ハードウェアに正しく接続されているかどうか確認します。 通常，ドライブは PCI スロット 2 のコネクタ・モジュール (Beeper の部品番号は 54-25133-01) にケーブル接続することができます。
3. ECU イメージを格納したフロッピィを挿入します。
4. プライマリ・コンソールから次のコマンドを入力します。

```
P08>>> SET ARC_ENABLE ON
P08>>> INITIALIZE
P08>>> RUNECU
```

5. ECU から要求される手順を実行します。
6. プライマリ・コンソールから次のコマンドを入力します。

```
P08>>> boot
$ @SYS$SYSTEM:SHUTDOWN
P08>>> SET ARC_ENABLE OFF
P08>>> INITIALIZE
P08>>> LPINIT
```

7. OpenVMS Galaxy をリブートします。

ECU には 2 つのバージョンがあり，1 つはグラフィックス端末で実行され， もう 1 つはキャラクタ・セル端末で実行されます。 どちらのバージョンもフロッピィに格納されており， コンソールはどちらのバージョンを実行するかを判断します。 OpenVMS Galaxy システムの場合， プライマリ・コンソールは常にキャラクタ・セル端末を装備したシリアル装置です。

ECU を実行しないと，OpenVMS は次のメッセージを表示します。

```
        %SYSTEM-I-NOCONFIGDATA, IRQ Configuration data for EISA
   slot xxx was not found, please run the ECU and reboot.
```

このメッセージを無視すると，システムはブートされますが， プラグイン EISA 装置は無視されます。

これまでの説明に従って OpenVMS Galaxy ハードウェアを構成し，設定した後， 次の手順を実行して OpenVMS Galaxy インスタンスをインストールし，ブートします。

6.3 ステップ 3: システム・ディスクの作成

インスタンスごとにシステム・ディスクを使用するのか， **または** クラスタ全体で共通のディスクで使用するのかを判断します。

OpenVMS バージョン 7.1-2 より以前のバージョンを稼働しているクラスタ・メンバのうち，Galaxy インスタンスと同じ VMS$OBJECTS.DAT ファイルを共用するクラスタ・メンバの場合は，新しい SECURITY.EXE が必要です。

6.4 ステップ 4: OpenVMS Alpha バージョン 7.3 のインストール

OpenVMS Galaxy ソフトウェアを実行するのに，特別なインストール手順は必要ありません。Galaxy 機能は基本オペレーティング・システムに組み込まれており， この章で説明するコンソール・コマンドとシステム・パラメータ値を使用して， 有効または無効に設定できます。

OpenVMS Alpha オペレーティング・システムのインストールの詳細については，『OpenVMS インストレーション・ガイド [翻訳版]』を参照してください。

OpenVMS Galaxy ライセンス情報は，『OpenVMS License Management Utility Manual』を参照してください。

6.5 ステップ 5: ファームウェアのアップグレード

AlphaServer 8200 で OpenVMS Galaxy 環境を構築するには， 各プロセッサ・モジュールでファームウェアのアップグレードが必要です。これらのモジュールを Galaxy 以外の構成で再び使用する場合は， 以前のファームウェアを再インストールする必要があります。 現在のファームウェア CD は必ず保管しておいてください。

使用する予定のすべてのプロセッサ・モジュールを取り付け，同時に更新しておけば，作業時間を短縮できます。AlphaServer 8200 では， すべてのプロセッサ・ボードで同じファームウェアを使用しなければなりません。後でボードをアップグレードしなければならない場合は，次の操作が必要です。

1. ファームウェア・リビジョン・レベルが異なるすべてのボードを取り外します。
2. 以前のボードを更新します。
3. 残りのボードを再び取り付けます。

ファームウェアをアップグレードするには，システムの電源をオンにし，非 Galaxy モードで稼働します (つまり，LP_COUNT コンソール環境変数を設定している場合，0 に設定する必要があります)。

コンソール環境変数を設定するには，次のコマンドを入力します。

```
P08>>> SET LP_COUNT 0
P08>>> INIT
```

ファームウェアをアップグレードするには，Alpha Systems Engineering から 提供されている標準コンソール・ファームウェア・アップデートを使用します。

6.6 ステップ 6: 環境変数の設定

すべてのプロセッサ・モジュールでファームウェアをアップグレードした後，次の例に示すように，Galaxy 固有の環境変数を作成できます。この例では， 2 つのインスタンス，4 つの CPU，1 GB の OpenVMS Galaxy コンピューティング環境を 構成しているものと仮定しています。

```
P08>>> create -nv lp_count          2
P08>>> create -nv lp_cpu_mask0      100
P08>>> create -nv lp_cpu_mask1      e00
P08>>> create -nv lp_io_mask0       100
P08>>> create -nv lp_io_mask1       80
P08>>> create -nv lp_mem_size0      10000000
P08>>> create -nv lp_mem_size1      10000000
P08>>> create -nv lp_shared_mem_size  20000000
P08>>> init
```

これらの変数を作成した後，コンソール SET コマンドを使用して， 変数を操作することができます。 これらの変数はプロセッサ 0 でのみ作成する必要があります。

ここでは各環境変数について詳しく説明します。

LP_COUNT number

この変数が 0 に設定されていると，システムは従来の SMP 構成だけをブートします。 Galaxy コンソール・モードは OFF になります。

この変数が 0 以外の値に設定されている場合は，Galaxy 機能が使用され， Galaxy 変数が解釈されます。LP_COUNT の正確な値は， コンソールが認識する Galaxy パーティションの数を表します。

LP_CPU_MASKn mask

このビット・マスクは，指定された Galaxy パーティション番号にどの CPU を最初に割り当てるかを指定します。 AlphaServer 8200 コンソールは， パーティション内の最初の偶数番号の CPU をイニシャル・インスタンスとして CPU 08 で始まるプライマリ CPU として選択します。リソースを割り当てる場合は，このことに注意してください (つまり，奇数番号だけの CPU をパーティションに割り当てないでください)。

LP_IO_MASKn mask

これらの変数は， スロット番号によって IO モジュールを各インスタンスに割り当てます。

- 100 はスロット 8 の I/O モジュールを表します。
- 80 はスロット 7 の I/O モジュールを表します。

AlphaServer 8200 の場合，ここに示した割り当てだけが有効です。

LP_MEM_SIZEn size

これらの変数は，指定されたインスタンスに対して特定の容量のプライベート・メモリを割り当てます。 システム内のメモリ容量と各インスタンスにとって必要な割り当てをもとに， 適切な値を使用してこれらの変数を作成することが必要です。 一般的に使用される値については，付録 B「メモリ・サイズ設定の共通値」 を参照してください。

また，この後の共用メモリ変数も参照してください。

LP_SHARED_MEM_SIZE size

この変数は，共用メモリとして使用するメモリを割り当てます。一般的に使われる値については，付録 B「メモリ・サイズ設定の共通値」 を参照してください。

注意:

共用メモリは 8 MB の倍数で割り当てなければならず， すべての値は 16 進バイトで表現されます。

使用する共用メモリの容量だけを定義でき， 他の LP_MEM_SIZE 変数は未定義のままにしておくことができます。このようにすると，コンソールは上位アドレス空間から共用メモリを割り当て， LP_COUNT 変数によって指定された数のパーティションに対して， 残りのメモリを等しく分割します。また， LP_MEM_SIZE 変数を使用して特定のパーティションにメモリを明示的に割り当て，他のパーティションのメモリ割り当てを未定義のままにした場合も，コンソールはメモリ・フラグメントを，明示的にメモリが割り当てられたパーティションおよび共用メモリに対して割り当て， 残りのメモリを分割して， 明示的にメモリが割り当てられていない残りのパーティションに割り当てます。

BOOTDEF_DEV 変数と BOOT_OSFLAGS 変数

初期インストールの後や，システム・クラッシュやオペレータによって要求されたリブートの後，システムをリブートしなければならない場合は，AUTOGEN が正しくリブートされるように，ブートの前に各 Galaxy コンソールでこれらの変数を設定する必要があります。

Galaxy の環境変数の例

```
P08>>> SHOW LP*

lp_count 2
lp_shared_mem_size 20000000   (512 MB)
lp_mem_size0 10000000 (256 MB)
lp_mem_size1 10000000 (256 MB)
lp_cpu_mask0 100 (CPU 0)
lp_cpu_mask1 e00 (CPUs 1-3)
lp_io_mask0 100 (I/O module in slot 8)
lp_io_mask1 80 (I/O module in slot 7)

P08>>>
```

6.7 ステップ 7: セカンダリ・コンソール装置の起動

KFE72-DA が Windows-NT に構成されていた場合は， ビデオ・ボードを検出しようとしますが，検出できないときはハングします。 OpenVMS Galaxy を構成する場合，このような状況が一般的に発生します。 操作モードを設定するには，コンソール・コマンドを使用します。

```
P08>>> SET CONSOLE SERIAL
```

セカンダリ・コンソールを初期化する前に， プライマリ・コンソールに対してこのコマンドを入力すると， 設定はセカンダリ・コンソール・ハードウェアに伝達されます。

イーサネット・ポートを使用する場合は，使用するメディアの種類と接続をコンソールに通知しなければなりません。 つまり，AUI，UDP，ツイスト・ペアのいずれを使用するのかを指定する必要があります。コンソールとオペレーティング・システムはどのメディアを使用するかを判断しますが，次のコマンドを使用すれば，特定のメディア・タイプを割り当てることができます。

```
P08>>> SHOW NETWORK

P08>>> SET EWA0_MODE TWISTED
```

最初のコマンドは，使用可能なネットワーク装置の一覧を表示します。2 番目のコマンドは，指定された装置 (この例では EWA0) に対してデフォルト・メディア・タイプを設定します。これはセカンダリ・コンソールを初期化する前に， すべてのイーサネット装置に対して実行しなければなりません。

コンソール・モードとネットワーク・メディア・タイプ(使用する場合)を設定した後，システムを再初期化して，現在の設定を保存します。 Galaxy パーティションをすでに定義している場合は， 初期化は後で実行しなければなりません。

システムを初期化できる場合は，次のコマンドを入力します。

```
P08>>> INIT
```

プライマリ・コンソールからの応答として，通常の電源投入時の自己診断テスト (POST) レポートが出力されます。 これには最大 2 分かかる可能性があります。Galaxy パーティションを適切に定義した場合は， プライマリ・パーティションに関連する I/O 装置だけが表示されます。

パーティションが定義されていることを確認するには，次のコマンドを入力します。

```
P08>>> SHOW DEVICE
```

または

```
P08>>> SHOW NETWORK
```

セカンダリ・コンソールを初期化するには，次のコマンドを入力します。

```
P08>>> LPINIT
```

コンソールに次の情報が表示されます。

```
Partition 0: Primary CPU = 0
Partition 1: Primary CPU = 2
Partition 0: Memory Base = 000000000   Size = 010000000
Partition 1: Memory Base = 010000000   Size = 010000000
Shared Memory Base = 020000000   Size = 010000000
LP Configuration Tree = 12c000
starting cpu 1 in Partition 1 at address 01000c001
starting cpu 2 in Partition 1 at address 01000c001
starting cpu 3 in Partition 1 at address 01000c001

P08>>>
```

このコマンドは **プライマリ** `Galaxy` **コンソール** から入力しなければなりません。 Galaxy パーティションが正しく定義されており，ハードウェア・リソースが正しく構成されている場合は，プライマリ・コンソールは各セカンダリ・パーティションに割り当てられているプロセッサを起動するはずです。 セカンダリ・コンソールは 2 分以内に初期化されます。

1 つ以上のコンソールの初期化が失敗した場合は，ハードウェアの取り付け，Galaxy パーティションの定義， ハードウェアの割り当てを二重にチェックする必要があります。

OpenVMS コンソールの制限事項とヒントの詳細については， 第12章 「OpenVMS Galaxy に関するヒントと手法」 を参照してください。

6.8 ステップ 8: OpenVMS Galaxy のブート

Galaxy ファームウェアを正しくインストールし，コンソールを構成した後， 次の方法で初期 Galaxy 環境をブートできます。

各 Galaxy インスタンスに対して，次のコマンドを入力します。

```
P08>>> B -FL 0,1 DKA100 // or whatever your boot device is.

SYSBOOT> SET GALAXY 1

SYSBOOT> CONTINUE
```

構成はこれで終了です。 これで OpenVMS Galaxy が構築されました。

第7章 AlphaServer 4100 システムでの OpenVMS Galaxy の構築

この章では，AlphaServer 4100 で OpenVMS Galaxy コンピューティング環境を構築するための要件と手順について説明します。

7.1 はじめに

AlphaServer 4100 で OpenVMS Galaxy を構築するには， 構成およびハードウェアに関して，次の要件を十分理解しておく必要があります。

最大 2 つのインスタンス

AlphaServer 4100 では，OpenVMS のインスンタスを最大 2 つ実行できます。

コンソール・ファームウェア

OpenVMS バージョン 7.3 CD-ROM にある AlphaServer 4100 コンソール・ファームウェアが必要です。

コンソール・コマンド

第5章 「AlphaServer 8400 システムでの OpenVMS Galaxy の構築」に示したコンソール・ヒントの他に，次のことにも注意してください。

- コンソール・コマンドは一度に 1 つのインスタンスで入力します。
- 最初のコンソールで入力されたコマンドが完了するまで，別のコンソールでコンソール・コマンドを入力しないでください。

AlphaServer 4100 のクロック

AlphaServer 4100 には 1 つのクロックがあります。OpenVMS Galaxy の場合， このことは 2 つのインスタンスを異なる時刻に実行できないことを意味します。 また，SET TIME コマンドは両方のインスタンスに影響します。しかし，かなり時間が経過するまで，このことは明らかにならない可能性があります。

コンソール・ポート

COM1 (上) はインスタンス 0 のコンソール・ポートです。 COM2 (下) はインスタンス 1 のコンソール・ポートです。

AlphaServer 8400 で OpenVMS Galaxy を構築する場合と異なり， 2 つ目のコンソール用に追加ハードウェアは必要ありません。 この目的で COM2 が使用されます。

CPU

CPU0 はインスタンス 0 のプライマリでなければなりません。 CPU1 はインスタンス 1 のプライマリでなければなりません。 CPU2 と 3 はオプションのセカンダリ CPU であり，マイグレードすることができます。

I/O アダプタ

下の 4 つの PCI スロットは IOD0 に属しています。 これはインスタンス 0 用の I/O アダプタです。 上の 4 つの PCI スロットは IOD1 に属しています。 これはインスタンス 1 用の I/O アダプタです。

ストレージ・コントローラ

KZPSA などのストレージ・コントローラが 2 つ必要です。 これらのコントローラは個別の Storageworks ボックスに収納することができ， SCSI クラスタとして稼働するために，同じボックスに収納することもできます。 各コントローラはそれぞれ IOD0 と IOD1 に接続されます。

ネットワーク・カード

各インスタンスでネットワーク・アクセスが必要な場合は， 各インスタンスに対してネットワーク・カード (DE500 など) が必要です。

カードは 1 枚ずつ，IOD0 と IOD1 に接続されます。

物理メモリ

AlphaServer 4100 の OpenVMS Galaxy では，メモリ・ホールがサポートされないため，OpenVMS Galaxy 環境用の物理メモリは連続していなければなりません。 AlphaServer 4100 でこのことを実現するには， 次のいずれかの条件を満たさなければなりません。

- すべてのメモリ・モジュールが同じサイズでなければなりません (たとえば 1 GB)。
- 2 種類のサイズがある場合は，1 つのモジュールだけを小さいサイズにすることができます。 大きいサイズのモジュールは番号の小さなスロットに挿入しなければなりません。

AlphaServer 4100 システムで OpenVMS Galaxy を構築するには， この後の節の操作を行います。

7.2 ステップ 1: AlphaServer 4100 構成を確かめる

SHOW CONFIG コマンドを使用して，OpenVMS Galaxy 環境を構築するために使用する AlphaServer 4100 が，7.1 項「はじめに」 で説明した要件を 満たしているかどうか確認します。

コンソール・プロンプトに対して次のコマンドを入力します。

```
P00>>> show config
```

コンソールに次の情報が表示されます。

```
Console G53_75  OpenVMS PALcode V1.19-16, UNIX PALcode V1.21-24

Module                          Type     Rev    Name
System Motherboard              0        0000   mthrbrd0
Memory  512 MB EDO              0        0000   mem0
Memory  256 MB EDO              0        0000   mem1
CPU (Uncached)                  0        0000   cpu0
CPU (Uncached)                  0        0000   cpu1
Bridge (IOD0/IOD1)              600      0021   iod0/iod1
PCI Motherboard                 8        0000   saddle0
CPU (Uncached)                  0        0000   cpu2
CPU (Uncached)                  0        0001   cpu3

Bus 0  iod0 (PCI0)
Slot   Option Name              Type     Rev    Name
1      PCEB                     4828086  0005   pceb0
4      DEC KZPSA                81011    0000   pks1
5      DECchip 21040-AA         21011    0023   tulip1

Bus 1  pceb0 (EISA Bridge connected to iod0, slot 1)
Slot   Option Name              Type     Rev    Name

Bus 0  iod1 (PCI1)
Slot   Option Name              Type     Rev    Name
1      NCR 53C810               11000    0002   ncr0
2      DECchip 21040-AA         21011    0024   tulip0
3      DEC KZPSA                81011    0000   pks0
```

7.3 ステップ 2: OpenVMS Alpha Version 7.3-2 をインストールする

OpenVMS Galaxy ソフトウェアを実行するために， 特別なインストール手順は必要ありません。 Galaxy 機能は基本オペレーティング・システムに組み込まれており， この章で後述するコンソール・コマンドとシステム・パラメータ値を使用して， 有効または無効に設定することができます。

AlphaServer 4100 が SCSI クラスタに属していない場合は，各インスタンスに対して 1 つずつ，2 つのシステム・ディスクに OpenVMS バージョン 7.3 をインストールしなければなりません。

AlphaServer 4100 がクラスタで共通のシステム・ディスクを持つ SCSI クラスタの一部である場合は，1 つのシステム・ディスクに OpenVMS バージョン 7.3 をインストールします。

OpenVMS Alpha オペレーティング・システムのインストールの詳細については，『OpenVMS インストレーション・ガイド [翻訳版]』を参照してください。

7.4 ステップ 3: ファームウェアをアップグレードする

ファームウェアをアップグレードするには，OpenVMS バージョン 7.3 の CD-ROM パッケージに含まれている「Alpha Systems Firmware Update Version 5.4 CD-ROM」を使用します。ファームウェアを実際にインストールする前に，パッケージに同梱されているリリース・ノートを参照してください。

7.5 ステップ 4: 環境変数を設定する

インスタンス 0 に対してプライマリ・コンソールを構成します。

CPU0 はインスタンス 0 のプライマリです。

Galaxy 環境変数を作成します。Galaxy 環境変数と各変数の一般的な値については，第5章「AlphaServer 8400 システムでの OpenVMS Galaxy の構築」を参照してください。

次の例は CPU 3 つと 256 MB + 192 MB + 64 MB に分割された 512 MB のメモリを装備した AlphaServer 4100 の場合の例です。

```
P00>>> create -nv lp_count            2
P00>>> create -nv lp_cpu_mask0        1
P00>>> create -nv lp_cpu_mask1        6
P00>>> create -nv lp_io_mask0         10
P00>>> create -nv lp_io_mask1         20
P00>>> create -nv lp_mem_size0        10000000
P00>>> create -nv lp_mem_size1        c000000
P00>>> create -nv lp_shared_mem_size  4000000
P00>>> set auto_action halt
```

CPU が 4 つあり，すべてのセカンダリ CPU をインスタンス 1 に割り当てる場合は，LP_CPU_MASK1 変数が E になります。2 つのインスタンスで CPU を分割する場合は，CPU 0 がインスタンス 0 のプライマリ CPU になり，CPU 1 がインスタンス 1 のプライマリ CPU にならなければなりません。

MEM_SIZE 変数は，システム構成とメモリの分割方法に応じて異なります。

- lp_io_mask0 は 10 に設定しなければなりません。
- lp_io_mask1 は 20 に設定しなければなりません。

コンソール環境変数 AUTO_ACTION は HALT に設定しなければなりません。これにより，システムはブートされず，Galaxy コマンドを入力できるようになります。

7.6 ステップ 5: システムを初期化し，コンソール装置を起動する

1. 次のコマンドを入力して，システムを初期化し，Galaxy ファームウェアを起動します。

   ```
   P00>>> init
   P00>>> galaxy
   ```

   自己診断テストを完了した後，Galaxy コマンドはインスタンス 1 でコンソールを起動します。

   Galaxy が初めて起動されると，次のような複数のメッセージが表示されます。

   ```
   CPU0 would not join
   ```

```
IOD0 and IOD1 did not pass the power-up self-test
```

これらのメッセージが表示されるのは，2 組の環境変数があり， Galaxy 変数は最初にインスタンス 1 に存在しないからです。

I/O バスが 2 つの Galaxy パーティション間で分割される場合は， 装置のポート名が変化することに注意してください。 たとえば，AlphaServer 4100 がシングル・システムの場合に， DKC300 として指定されるディスクは， OpenVMS Galaxy のパーティション 0 として構成した場合は，DKA300 になります。

2. インスタンス 1 のコンソールを構成します。

```
P01>>> create -nv lp_cpu_mask0        1
P01>>> create -nv lp_cpu_mask1        6
P01>>> create -nv lp_io_mask0         10
P01>>> create -nv lp_io_mask1         20
P01>>> create -nv lp_mem_size0        10000000
P01>>> create -nv lp_mem_size1        c000000
P01>>> create -nv lp_count       2
P01>>> create -nv lp_shared_mem_size  4000000
P01>>> set auto_action halt
```

3. 次のコマンドを入力して，システムを初期化し，Galaxy ファームウェアを再起動します。

```
P00>>> init
```

コンソールに次の確認メッセージが表示されたら，Y と入力します。

```
Do you REALLY want to reset the Galaxy (Y/N)
```

4. システム・ルート，ブート装置，他の関連変数を構成します。

次の設定名は OpenVMS Engineering システムの場合の例です。 それぞれの環境の要件に適合するように，これらの変数は適宜変更してください。

```
P00>>> set boot_osflags 12,0
P00>>> set bootdef_dev dka0
P00>>> set boot_reset off             !!! must be OFF !!!
P00>>> set ewa0_mode twisted

P01>>> set boot_osflags 11,0
P01>>> set bootdef_dev dkb200
P01>>> set boot_reset off             !!! must be OFF !!!
P01>>> set ewa0_mode twisted
```

5. 次のように，インスタンス 1 をブートします。

```
P01>>> boot
```

インスタンス 1 がブートされた後，システム・アカウントにログインし，SYS$SYSTEM:MODPARAMS.DAT ファイルに次の行を挿入します。

```
GALAXY=1
```

SCS ノードと SCS システム ID の行が正しいことを確認してください。 次のように AUTOGEN を実行して，インスタンス 1 を Galaxy メンバとして構成し， システムを停止したままの状態にします。

```
$ @SYS$UPDATE:AUTOGEN GETDATA SHUTDOWN INITIAL
```

6. 次の手順を実行して，インスタンス 0 をブートします。

```
P00>>> boot
```

インスタンス 0 がブートされた後，システム・アカウントにログインし，SYS$SYSTEM:MODPARAMS.DAT ファイルに次の行を追加します。

```
GALAXY=1
```

SCS ノードと SCS システム ID の行が正しいことを確認してください。 次の手順で AUTOGEN を実行して，インスタンス 0 を Galaxy メンバとして構成し， システムを停止したままの状態にします。

```
$ @SYS$UPDATE:AUTOGEN GETDATA SHUTDOWN INITIAL
```

7. 初期化またはシステムのパワー・サイクルで自動的に起動されるように， Galaxy を準備します。 両方のインスタンスで AUTO_ACTION 環境変数を RESTART に設定します。

```
P00>>> set auto_action restart

P01>>> set auto_action restart
```

8. プライマリ・コンソールから次のコマンドを入力して，Galaxy を再び初期化します。

```
P00>>> init
```

コンソールに次の確認メッセージが表示されたら，Y と入力します。

```
Do you REALLY want to reset the Galaxy (Y/N)
```

また，システムの電源をいったんオフにした後，オンにすることもできます。このようにすると， 両方のインスタンスで Galaxy が自動的にブートストラップされます。

操作はこれで終了です。OpenVMS Galaxy が構築されました。

第8章 AlphaServer ES40 システムでの OpenVMS Galaxy の構築

この章では，AlphaServer ES40 システムで OpenVMS Galaxy コンピューティング環境を 構築するための要件と手順について説明します。

この章には，当初は OpenVMS Alpha VMS721_DS20E_ES40 修正キットに含まれていた手順の改訂版が含まれています。

AlphaServer ES40 システムで OpenVMS Galaxy を構築するには，次の手順に従います。

1. 8.1 項「コンピューティング環境を構築する前に」に記載された，構成とハードウェアの要件を読みます。
2. ステップ 1 からステップ 5 までの手順を実行します。

8.1 コンピューティング環境を構築する前に

AlphaServer ES40 の構成およびハードウェアに関して， 次の要件を十分理解しておく必要があります。

最大 2 つのインスタンス

AlphaServer ES40 では，OpenVMS のインスンタスを最大 2 つ実行できます。

コンソール・ファームウェア

AlphaServer ES40 システムで OpenVMS Galaxy 環境を構築するには， 次の場所から V6.2 コンソール・ファームウェアの最新バージョンを ダウンロードしなければなりません。

http://ftp.digital.com/pub/DEC/Alpha/firmware/

AlphaServer ES40 のクロック

AlphaServer ES40 には 1 つのクロックがあります。OpenVMS Galaxy の場合， これは 2 つのインスタンスを異なる時刻に実行できないことを意味します。 また，SET TIME コマンドは両方のインスタンスに影響します。しかし，かなり時間が経過するまで，これが明らかにならない可能性があります。

コンソール・ポート

ラック・マウント・システムの場合:
COM1 (下) はインスタンス 0 のコンソール・ポートです。
COM2 (上) はインスタンス 1 のコンソール・ポートです。

ペデスタル・システムの場合:
COM1 (左) はインスタンス 0 のコンソール・ポートです。
COM2 (右) はインスタンス 1 のコンソール・ポートです。

AlphaServer 8400 で OpenVMS Galaxy を構築する場合と異なり， 2 つ目のコンソール用に追加ハードウェアは必要ありません。 この目的で COM2 が使用されます。

CPU

CPU0 はインスタンス 0 のプライマリでなければなりません。
CPU1 はインスタンス 1 のプライマリでなければなりません。
CPU2 と 3 はオプションのセカンダリ CPU であり，マイグレードすることができます。

AlphaServer ES40 における CPU 環境変数の設定例については， 8.5 項「ステップ 4: 環境変数の設定」 を参照してください。

I/O アダプタ

ラックマウント・システムの場合:
PCI hose 0 (PCI0) はインスタンス 0 に属しています (上の 4 つの PCI スロット)。
PCI hose 1 (PCI1) はインスタンス 1 に属しています (下の 6 つの PCI スロット)。

ペデスタル・システムの場合:

PCI hose 0 (PCI0) はインスタンス 0 に属しています (右側のスロット)
PCI hose 1 (PCI1) はインスタンス 1 に属しています (左側のスロット)

PCI0 には内蔵型の ISA コントローラがあります。

I/O アダプタの構成例については，8.2 項「ステップ 1: AlphaServer ES40 の構成を確かめる」 を参照してください。

ストレージ・コントローラ

インスタンスごとに 1 つのストレージ・コントローラ (KZPSA など) が必要です。 それぞれのインスタンスに対して，コントローラは個別の StorageWork ボックスに収納することができ，SCSI クラスタとして稼働するために， 同じボックスに収納することもできます。

ネットワーク・カード

各インスタンスでネットワーク・アクセスが必要な場合は， 各インスタンスに対してネットワーク・カード (DE600 など) が必要です。

カードは 1 枚ずつ，PCI0 と PCI0 に接続されます。

メモリ分割に関する制限事項

プライベート・メモリは 64MB 境界から開始しなければなりません。

共用メモリは 8MB 境界から開始しなければなりません。

インスタンス 0 は 64MB の倍数でなければなりません。

8.2 ステップ 1: AlphaServer ES40 の構成を確かめる

SHOW CONFIG コマンドを使用して， OpenVMS Galaxy 環境を構築するために使用する AlphaServer ES40 が， 8.1 項「コンピューティング環境を構築する前に」 で説明した要件を満たしているかどうか確認します。

コンソール・プロンプトに対して次のコマンドを入力します。

```
P00>>> show config
```

コンソールには次の例のような情報が表示されます。

```
Firmware
SRM Console:    X5.6-2323
ARC Console:    v5.70
PALcode:        OpenVMS PALcode V1.61-2, Tru64 UNIX PALcode V1.54-2
Serial Rom:     V2.2-F
RMC Rom:        V1.0
RMC Flash Rom:  T2.0

Processors
CPU 0           Alpha 21264-4 500 MHz  4MB Bcache
CPU 1           Alpha 21264-4 500 MHz  4MB Bcache
CPU 2           Alpha 21264-4 500 MHz  4MB Bcache
CPU 3           Alpha 21264-4 500 MHz  4MB Bcache

Core Logic
Cchip           DECchip 21272-CA Rev 9(C4)
Dchip           DECchip 21272-DA Rev 2
Pchip 0         DECchip 21272-EA Rev 2
Pchip 1         DECchip 21272-EA Rev 2
TIG             Rev 10

Memory
  Array       Size       Base Address    Intlv Mode
---------  ----------  ----------------  ----------
    0       4096Mb     0000000000000000    2-Way
    1       4096Mb     0000000100000000    2-Way
    2       1024Mb     0000000200000000    2-Way
    3       1024Mb     0000000240000000    2-Way

     10240 MB of System Memory
```

```
Slot   Option                  Hose 0, Bus 0, PCI
  1    DAPCA-FA ATM622 MMF
  2    DECchip 21152-AA                              Bridge to Bus 2, PCI
  3    DEC PCI FDDI            fwb0.0.0.3.0          00-00-F8-BD-C6-5C
  4    DEC PowerStorm
  7    Acer Labs M1543C                              Bridge to Bus 1, ISA
 15    Acer Labs M1543C IDE    dqa.0.0.15.0
                               dqb.0.1.15.0
                               dqa0.0.0.15.0         TOSHIBA CD-ROM XM-6302B
 19    Acer Labs M1543C USB

       Option                  Hose 0, Bus 1, ISA
       Floppy                  dva0.0.0.1000.0

Slot   Option                  Hose 0, Bus 2, PCI
  0    NCR 53C875              pkd0.7.0.2000.0       SCSI Bus ID 7
  1    NCR 53C875              pke0.7.0.2001.0       SCSI Bus ID 7
                               dke100.1.0.2001.0     RZ1BB-CS
                               dke200.2.0.2001.0     RZ1BB-CS
                               dke300.3.0.2001.0     RZ1CB-CS
                               dke400.4.0.2001.0     RZ1CB-CS
  2    DE500-AA Network Con    ewa0.0.0.2002.0       00-06-2B-00-0A-58

Slot   Option                  Hose 1, Bus 0, PCI
  1    NCR 53C895              pka0.7.0.1.1          SCSI Bus ID 7
                               dka100.1.0.1.1        RZ2CA-LA
                               dka300.3.0.1.1        RZ2CA-LA
  2    Fore ATM 155/622 Ada
  3    DEC PCI FDDI            fwa0.0.0.3.1          00-00-F8-45-B2-CE
  4    QLogic ISP10x0          pkb0.7.0.4.1          SCSI Bus ID 7
                               dkb100.1.0.4.1        HSZ50-AX
                               dkb101.1.0.4.1        HSZ50-AX
                               dkb200.2.0.4.1        HSZ50-AX
                               dkb201.2.0.4.1        HSZ50-AX
                               dkb202.2.0.4.1        HSZ50-AX
  5    QLogic ISP10x0          pkc0.7.0.5.1          SCSI Bus ID 7
                               dkc100.1.0.5.1        RZ1CB-CS
                               dkc200.2.0.5.1        RZ1CB-CS
                               dkc300.3.0.5.1        RZ1CB-CS
                               dkc400.4.0.5.1        RZ1CB-CS
  6    DECchip 21154-AA                              Bridge to Bus 2, PCI

Slot   Option                  Hose 1, Bus 2, PCI
  4    DE602-AA                eia0.0.0.2004.1       00-08-C7-91-0A-AA
  5    DE602-AA                eib0.0.0.2005.1       00-08-C7-91-0A-AB
  6    DE602-TA                eic0.0.0.2006.1       00-08-C7-66-80-9E
  7    DE602-TA                eid0.0.0.2007.1       00-08-C7-66-80-5E
```

8.3 ステップ 2: OpenVMS Alpha バージョン 7.3-2 をインストールする

OpenVMS Galaxy ソフトウェアを実行するために， 特別なインストール手順は必要ありません。 Galaxy 機能は基本オペレーティング・システムに組み込まれており， この章で後述するコンソール・コマンドとシステム・パラメータ値を使用して， 有効または無効に設定することができます。

AlphaServer ES40 が SCSI クラスタに属していない場合は，各インスタンスに対して 1 つずつ，2 つのシステム・ディスクに OpenVMS バージョン 7.3-2 をインストールしなければなりません。

AlphaServer ES40 がクラスタで共通のシステム・ディスクを持つ SCSI クラスタの一部である場合は， 1 つのシステム・ディスクに OpenVMS バージョン 7.3-2 をインストールします。

OpenVMS Alpha オペレーティング・システムのインストールの詳細については，『OpenVMS インストレーション・ガイド [翻訳版]』を参照してください。

8.4 ステップ 3: ファームウェアをアップグレードする

ファームウェアのアップグレードには，次の手順のいずれかを使います。

AlphaServer ES40 にアクセス可能である，MOP が有効になったサーバの MOM$SYSTEM にファームウェア・ファイルをコピーします。 コンソールから次のコマンドを入力します。

```
P00>>> boot -fl 0,0 ewa0 -fi {firmware filename}
UPD> update srm*
power-cycle system
```

あるいは，次のコマンドを使います。

```
P00>>> BOOT -FLAGS 0,A0 cd_device_name
.
.
.
Bootfile: {firmware filename}
.
.
.
```

8.5 ステップ 4: 環境変数の設定

インスタンス 0 に対してプライマリ・コンソールを構成します。

CPU0 はインスタンス 0 のプライマリです。 CPU1 はインスタンス 1 のプライマリです。

次の例は CPU 3 つと 256 MB + 192 MB + 64 MB に分割された 512 MB のメモリを装備した AlphaServer ES40 の場合の例です。

```
P00>>> set  lp_count            2
P00>>> set  lp_cpu_mask0        1
P00>>> set  lp_cpu_mask1        6
P00>>> set  lp_io_mask0         1
P00>>> set  lp_io_mask1         2
P00>>> set  lp_mem_size0        10000000
P00>>> set  lp_mem_size1        c000000
P00>>> set  lp_shared_mem_size  4000000
P00>>> set  console_memory_allocation new
P00>>> set auto_action halt
```

CPU が 4 つあり，すべてのセカンダリ CPU をインスタンス 1 に割り当てる場合は，LP_CPU_MASK1 変数が E になります。2 つのインスタンスで CPU を分割する場合は， CPU 0 がインスタンス 0 のプライマリ CPU になり， CPU 1 がインスタンス 1 のプライマリ CPU にならなければなりません。

次の例は，セカンダリ CPU 2 をプライマリ CPU 0 に割り当て， セカンダリ CPU 3 をプライマリ CPU 1 に割り当てる LP_CPU_MASK 値を示しています。

```
>>> set lp_cpu_mask0 5
>>> set lp_cpu_mask1 A


CPU Selection                          LP_CPU_MASK

0(primary partition 0)                 2^0 =   1
1(primary partition 1)                 2^1 =   2
2(secondary)                           2^2 =   4
3(secondary)                           2^3 =   8
```

MEM_SIZE 変数は，システム構成とメモリの分割方法に応じて異なります。

- lp_io_mask0 は 1 に設定しなければなりません。

- lp_io_mask1 は 2 に設定しなければなりません。

コンソール環境変数 AUTO_ACTION は HALT に設定しなければなりません。 これにより，システムはブートされず，LPINIT コマンドを入力できるようになります。

8.6 ステップ 5: システムを初期化し，コンソール装置を起動する

1. 次のコマンドを入力して，システムを初期化し，Galaxy ファームウェアを起動します。

```
P00>>> init      ! initialize the system
P00>>> lpinit    ! start firmware
```

自己診断テストを完了した後， Galaxy コマンドはインスタンス 1 でコンソールを起動します。

I/O バスが 2 つの Galaxy パーティション間で分割される場合は， 装置のポート名が変化することに注意してください。 たとえば，AlphaServer ES40 がシングル・システムの場合に， DKC300 として指定されるディスクは， OpenVMS Galaxy のパーティション 0 として構成した場合は，DKA300 になります。

2. インスタンス 1 のコンソールを構成します。
3. システム・ルート，ブート装置，他の関連変数を構成します。

次の設定例は OpenVMS Engineering システムの場合の例です。 それぞれの環境の要件に適合するように，これらの変数は適宜変更してください。

```
      Instance 0
P00>>> set boot_osflags 12,0
P00>>> set bootdef_dev dka0
P00>>> set boot_reset off              !!! must be OFF !!!
P00>>> set ewa0_mode twisted


      Instance 1
P01>>> set boot_osflags 11,0
P01>>> set bootdef_dev dkb200
P01>>> set boot_reset off              !!! must be OFF !!!
P01>>> set ewa0_mode twisted
```

4. 次のように，インスタンス 1 をブートします。

```
P01>>> boot
```

インスタンス 1 がブートされた後，システム・アカウントにログインし，SYS$SYSTEM:MODPARAMS.DAT ファイルに次の行を挿入します。

```
GALAXY=1
```

SCSNODE パラメータと SCSSYSTEMID SYSGEN パラメータが正しいことを確認してください。次のように AUTOGEN を実行して， インスタンス 1 を Galaxy メンバとして構成し， システムを停止したままの状態にします。

```
$ @SYS$UPDATE:AUTOGEN GETDATA SHUTDOWN INITIAL
```

5. 次のように，インスタンス 0 をブートします。

```
P00>>> boot
```

インスタンス 0 がブートされた後，システム・アカウントにログインし，SYS$SYSTEM:MODPARAMS.DAT ファイルに次の行を追加します。

```
GALAXY=1
```

SCSNODE パラメータと SCSSYSTEMID SYSGEN パラメータが正しいことを確認してください。次のように AUTOGEN を実行して， インスタンス 0 を Galaxy メンバとして構成し， システムを停止したままの状態にします。

```
$ @SYS$UPDATE:AUTOGEN GETDATA SHUTDOWN INITIAL
```

6. 初期化またはシステムの電源投入で自動的に起動されるように，Galaxy を準備します。両方のインスタンスで AUTO_ACTION 環境変数を RESTART に設定します。

```
P00>>> set auto_action restart

P01>>> set auto_action restart
```

7. プライマリ・コンソールから次のコマンドを入力して，Galaxy を再び初期化します。

```
P00>>> init
```

コンソールに次の確認メッセージが表示されたら，Y と入力します。

```
Do you REALLY want to reset all partitions? (Y/N)
```

また，システムの電源をいったんオフにした後，オンにすることもできます。このようにすると， 両方のインスタンスで Galaxy が自動的にブートストラップされます。

操作はこれで終了です。 AlphaServer ES40 システムに OpenVMS Galaxy が構築されました。

第9章 AlphaServer GS80/160/320 システムでの OpenVMS Galaxy の構築

この章では，AlphaServer GS80/160/320 システムで OpenVMS Galaxy コンピューティング環境を構築するプロセスについて説明します。

9.1 ステップ 1: 構成の選択とハードウェア要件の判断

OpenVMS Alpha バージョン 7.3 は，AlphaServer GS160 システムにおいて， 次の最大構成をサポートします。

- 4 インスタンス
- 4 QBB
- 16 CPU
- 128 GB メモリ

規則:

- パーティションごとに標準 COM1 UART コンソール・ラインを持たなければならない。
- パーティションごとに PCI ドローワを持たなければならない。
- パーティションごとに 1 つの I/O モジュールを持たなければならない。
- パーティションごとに少なくとも 1 つの CPU モジュールを持たなければならない。
- パーティションごとに少なくとも 1 つのメモリ・モジュールを持たなければならない。

9.2 ステップ 2: ハードウェアの設定

構成に必要なハードウェアを取得したら，ハードウェア・マニュアルに記述された手順を適宜参照して取り付けてください。

9.3 ステップ 3: システム・ディスクの作成

インスタンスごとにシステム・ディスクを使用するのか，**または**クラスタ全体で共通のディスクで使用するのかを判断します。

OpenVMS バージョン 7.1-2 より以前のバージョンを稼働しているクラスタ・メンバのうち，Galaxy インスタンスと同じ VMS$OBJECTS.DAT ファイルを共用するクラスタ・メンバの場合は，新しい SECURITY.EXE ファイルが必要です。

9.4 ステップ 4: OpenVMS Alpha バージョン 7.3-2 のインストール

OpenVMS Galaxy ソフトウェアを実行するのに，特別なインストール手順は必要ありません。Galaxy 機能は基本オペレーティング・システムに組み込まれており， この章で説明するコンソール・コマンドとシステム・パラメータ値を使用して， 有効または無効に設定できます。

OpenVMS Alpha オペレーティング・システムのインストールの詳細については，『OpenVMS インストレーション・ガイド [翻訳版]』を参照してください。

9.4.1 OpenVMS Galaxy ライセンス情報

Galaxy 環境では，システム・スタートアップ中およびインスタンス間の CPU 再割り当てが発生するたびに，OPENVMS-GALAXY ライセンス・ユニットがチェックされます。

CPU 起動時に CPU をサポートする OPENVMS-GALAXY ライセンス・ユニットが不十分な場合，CPU はインスタンスの構成セットには存在しますが，停止されます。 後で適切なライセンス・ユニットをロードすると，停止した CPU をシステムの実行中に開始できます。これは，CPU が 1 つ以上の場合に当てはまります。

9.5 ステップ 5: 環境変数の設定

オペレーティング・システムをインストールした後，この項の例で示すように， Galaxy 固有の環境変数を作成できます。

9.5.1 AlphaServer GS160 の例

この AlphaServer GS160 での例は，次を装備した OpenVMS Galaxy コンピューティング環境を構成しているものと仮定しています。

- 4 インスタンス
- 4 QBB
- 16 CPU
- 32 GB メモリ

```
P00>>> show lp*

lp_count              4
lp_cpu_mask0          000F
lp_cpu_mask1          00F0
lp_cpu_mask2          0F00
lp_cpu_mask3          F000
lp_cpu_mask4          0
lp_cpu_mask5          0
lp_cpu_mask6          0
lp_cpu_mask7          0
lp_error_target       0
lp_io_mask0           1
lp_io_mask1           2
lp_io_mask2           4
lp_io_mask3           8
lp_io_mask4           0
lp_io_mask5           0
lp_io_mask6           0
lp_io_mask7           0
lp_mem_size0          0=4gb
lp_mem_size1          1=4gb
lp_mem_size2          2=4gb
lp_mem_size3          3=4gb
lp_mem_size4          0
lp_mem_size5          0
lp_mem_size6          0
lp_mem_size7          0
lp_shared_mem_size    16gb

P00>>> lpinit
```

9.5.2 AlphaServer GS320 の例

この AlphaServer GS320 システムでの例は，次を装備した OpenVMS Galaxy コンピューティング環境を構成することを想定しています。

- 4 インスタンス
- 8 QBB
- 32 CPU
- 32 GB メモリ

```
P00>>> show lp*

lp_count              4
lp_cpu_mask0          000F000F
lp_cpu_mask1          00F000F0
```

```
lp_cpu_mask2          0F000F00
lp_cpu_mask3          F000F000
lp_cpu_mask4          0
lp_cpu_mask5          0

lp_cpu_mask6          0
lp_cpu_mask7          0
lp_error_target       0
lp_io_mask0           11
lp_io_mask1           22
lp_io_mask2           44
lp_io_mask3           88
lp_io_mask4           0
lp_io_mask5           0
lp_io_mask6           0
lp_io_mask7           0
lp_mem_size0          0=2gb, 4=2gb
lp_mem_size1          1=2gb, 5=2gb
lp_mem_size2          2=2gb, 6=2gb
lp_mem_size3          3=2gb, 7=2gb
lp_mem_size4          0
lp_mem_size5          0
lp_mem_size6          0
lp_mem_size7          0
lp_shared_mem_size    16gb

P00>>> lpinit
```

9.5.3 環境変数の説明

この項ではそれぞれの環境変数について説明します。使用法の詳細については，『AlphaServer GS80/160/320 Firmware Reference Manual』を参照してください。

LP_COUNT number

この変数が 0 に設定されていると，システムは従来の SMP 構成だけをブートします。 Galaxy コンソール・モードは OFF になります。

この変数が 0 以外の値に設定されている場合は，Galaxy 機能が使用され， Galaxy 変数が解釈されます。LP_COUNT の正確な値は， コンソールが作成する Galaxy パーティションの数を表します。

3 つのパーティションにリソースを割り当て，LP_COUNT を 2 に設定した場合， 残りのリソースは割り当てられないままになります。

LP_CPU_MASKn mask

このビット・マスクは，指定された Galaxy パーティション番号にどの CPU を最初に割り当てるかを指定します。 AlphaServer GS160 コンソールは， パーティション内のセルフ・テストを通過した最初の CPU をプライマリ CPU として選択します。

LP_ERROR_TARGET

新しい AlphaServer GS シリーズは， LP_ERROR_TARGET という新しい Galaxy 環境変数を導入しました。 変数の値は，システム・エラーが最初に報告される Galaxy インスタンスの番号です。 他の Galaxy プラットフォームとは異なり， システムの修正可能なエラー，修正不可能なエラー，システム・イベント・エラーは， すべて単一のインスタンスに送られます。 オペレーティング・システムがこのターゲットを変更できるので，変数の値は， システムが最初にパーティション分割されたときのターゲットを表しています。

この目的はシステム・エラーを単一のインスタンスに隔離して， エラーが Galaxy 全体をダウンさせないことです。 エラーのターゲット・インスタンスは，エラーを受け取ったときに，エラーを被ったその単一のインスタンスをリモートからクラッシュさせても安全であるかどうかを判断します。この場合，GLXRMTMCHK のバグチェック・コードが使用されます。エラーに関係するエラー・ログ情報は， 必ずしもエラーを被ったインスタンスではなく， エラー・ターゲット・インスタンスに対するものであることに注意してください。

エラー・ターゲット・インスタンスは，可能な限りユーザが環境変数で指定したインスタンスのままになりますが，ソフトウェアはインスタンスを監視し，必要に応じてエラー・ターゲットを変更します。

LP_IO_MASKn mask

これらの変数は，QBB 番号によって I/O モジュールを各インスタンスに割り当てます。

マスク値	QBB 番号
1	QBB 0
2	QBB 1
4	QBB 2
8	QBB 3

n にはパーティション番号 (0～7) を指定します。mask 値は，(I/O ライザーを含む) どの QBB がパーティションに含まれるのかを示すバイナリ・マスクです。

LP_MEM_SIZEn size

これらの変数は，指定されたインスタンスに対して特定の容量のプライベート・メモリを割り当てます。システム内のメモリ容量と各インスタンスにとって必要な割り当てをもとに，適切な値を使用してこれらの変数を作成することが必要です。

使用する共用メモリの容量だけを定義でき，他の LP_MEM_SIZE 変数は未定義のままにしておくことができます。このようにすると，コンソールは上位アドレス空間から共用メモリを割り当て，LP_COUNT 変数によって指定された数のパーティションに対して，残りのメモリを等しく分割します。また，LP_MEM_SIZE 変数を使用して特定のパーティションにメモリを明示的に割り当て，他のパーティションのメモリ割り当てを未定義のままにした場合も，コンソールはメモリ・フラグメントを，明示的にメモリが割り当てられたパーティションおよび共用メモリに対して割り当て，残りのメモリを分割して，明示的にメモリが割り当てられていない残りのパーティションに割り当てます。

次の例を参照してください。

```
lp_mem_size0  0=2gb, 1=2gb
```

注意:

インスタンス中に CPU のない QBB からのインスタンスに，プライベート・メモリを割り当てないでください。

たとえば，LP_CPU_MASK0 が FF の場合，QBB 0 と 1 からのインスタンス 0 に対してのみプライベート・メモリを割り当ててください。

この変数の使用法の詳細については，『AlphaServer GS80/160/320 Firmware Reference Manual』を参照してください。

LP_SHARED_MEM_SIZE size

この変数は，共用メモリとして使用するメモリを割り当てます。次の例を参照してください。

```
lp_shared_mem_size      16gb
```

共用メモリは 8 MB の倍数で割り当てなければなりません。

この変数の使用法の詳細については，『AlphaServer GS80/160/320 Firmware Reference Manual』を参照してください。

BOOTDEF_DEV 変数と BOOT_OSFLAGS 変数

初期インストールの後や，システムの障害やオペレータによって要求されたリブートの後，AUTOGEN が正しくリブートされるように，ブートの前に各 Galaxy コンソールでこれらの変数を設定する必要があります。

9.6 ステップ 6: セカンダリ・コンソール装置の起動

イーサネット・ポートを使用する場合は，使用するメディアの種類と接続をコンソールに通知しなければなりません。つまり， AUI，UDP，ツイスト・ペアのいずれを使用するのかを指定する必要があります。コンソールとオペレーティング・システムはどのメディアを使用するかを判断しますが，次のコマンドを入力すれば，特定のメディア・タイプを割り当てることができます。

```
P00>>> SHOW NETWORK

P00>>> SET EWA0_MODE TWISTED
```

最初のコマンドは，使用可能なネットワーク装置の一覧を表示します。2 番目のコマンドは，指定された装置 (この例では EWA0) に対してデフォルト・メディア・タイプを設定します。これはセカンダリ・コンソールを初期化する前に，すべてのイーサネット装置に対して実行しなければなりません。

9.7 ステップ 7: セカンダリ・コンソールの初期化

Galaxy 変数を決定した後は， セカンダリ・コンソールを初期化するために次のコマンドを入力します。

```
P00>>> LPINIT
```

コンソールに次の情報が表示されます。

```
P00>>>lpinit
lp_count = 2
lp_mem_size0 = 1800 (6 GB)
CPU 0 chosen as primary CPU for partition 0
lp_mem_size1 = 1800 (6 GB)
CPU 4 chosen as primary CPU for partition 1
lp_shared_mem_size = 1000 (4 GB)
initializing shared memory
partitioning system
QBB 0 PCA 0 Target 0 Interrupt Count = 2
QBB 0 PCA 0 Target 0 Interrupt CPU = 0
Interrupt Enable = 000011110000d05a
Sent Interrupts = 0000100000000010
Enabled Sent Interrupts = 0000100000000010
Acknowledging Sent Interrupt 0000000000000010 for CPU 0
QBB 0 PCA 0 Target 0 Interrupt Count = 1
QBB 0 PCA 0 Target 0 Interrupt CPU = 0
Interrupt Enable = 000011110000d05a
Sent Interrupts = 0000100000000000 Enabled Sent Interrupts = 0000100000000000
Acknowledging Sent Interrupt 0000100000000000 for CPU 0


OpenVMS PALcode V1.80-1, Tru64 UNIX PALcode V1.74-1

system = QBB 0 1 2 3         + HS                              (Hard Partition 0)
 QBB 0 = CPU 0 1 2 3 + Mem 0       + Dir + IOP + PCA 0 1     + GP  (Hard QBB 0)
 QBB 1 = CPU 0 1 2 3 + Mem 0       + Dir + IOP + PCA 0 1     + GP  (Hard QBB 1)
 QBB 2 = CPU 0 1 2 3 + Mem 0       + Dir + IOP + PCA         + GP  (Hard QBB 4)
 QBB 3 = CPU 0 1 2 3 + Mem 0       + Dir + IOP + PCA         + GP  (Hard QBB 5)
partition 0
 CPU 0 1 2 3 8 9 10 11
 IOP 0 2
 private memory size is 6 GB
 shared memory size is 4 GB
micro firmware version is T5.4
shared RAM version is 1.4
```

```
hose 0 has a standard I/O module
starting console on CPU 0
QBB 0 memory, 4 GB
QBB 1 memory, 4 GB
QBB 2 memory, 4 GB
QBB 3 memory, 4 GB
total memory, 16 GB
probing hose 0, PCI
probing PCI-to-ISA bridge, bus 1

bus 1, slot 0 -- dva -- Floppy
bus 0, slot 1 -- pka -- QLogic ISP10x0
bus 0, slot 2 -- pkb -- QLogic ISP10x0
bus 0, slot 3 -- ewa -- DE500-BA Network Controller
bus 0, slot 15 -- dqa -- Acer Labs M1543C IDE
probing hose 1, PCI
probing hose 2, PCI
bus 0, slot 1 -- fwa -- DEC PCI FDDI
probing hose 3, PCI
starting console on CPU 1
starting console on CPU 2
starting console on CPU 3
starting console on CPU 8
starting console on CPU 9
starting console on CPU 10
starting console on CPU 11
initializing GCT/FRU at 1fa000
initializing pka pkb ewa fwa dqa
Testing the System
Testing the Disks (read only)
Testing the Network
AlphaServer Console X5.8-2842, built on Apr  6 2000 at 01:43:42
P00>>>
```

このコマンドは **プライマリ** `Galaxy` **コンソール** から入力しなければなりません。 Galaxy パーティションが正しく定義されており，ハードウェア・リソースが正しく構成されている場合は， 各インスタンスのプライマリ CPU が起動しているはずです。

1 つ以上のコンソールの初期化が失敗した場合は，ハードウェアの取り付け，Galaxy パーティションの定義， ハードウェアの割り当てを二重にチェックする必要があります。

9.8 ステップ 8: OpenVMS Galaxy のブート

Galaxy ファームウェアを正しくインストールし，コンソールを構成した後， 次の方法で初期 Galaxy 環境をブートできます。

各 Galaxy インスタンスに対して，次の操作を実行します。

```
P00>>> B -FL 0,1 DKA100 // or whatever your boot device is.

SYSBOOT> SET GALAXY 1

SYSBOOT> CONTINUE
```

構成はこれで終了です。これで OpenVMS Galaxy が構築されました。

第10章　AlphaServer ES47/ES80/GS1280 システムでの OpenVMS Galaxy の構築

この章では，AlphaServer ES47/ES80/GS1280 システムで OpenVMS Galaxy コンピューティング環境を構築するプロセスについて説明します。

注意:

ES47/ES80/GS1280 シリーズでハード・パーティションを利用するには，最低でも V6.6 のコンソール・セットが必要です。このコンソールは，第1章「パーティションによる負荷の管理」に記載されている AlphaServer ファームウェアの Web サイトから入手できます。

10.1 ステップ 1: 構成の選択とハードウェア要件の判断

ES47/ES80/GS1280 では，以前の AlphaServer プラットフォームと違い，Galaxy パーティションは SRM コンソールの外部に設定され，設定が必要な環境変数はありません。 OpenVMS Alpha バージョン 7.3-2 では，各 AlphaServer システムに対し，表 10-1 ES47/ES80/GS1280 の構成 に示す最大構成がサポートされます。

それぞれのパーティションで共用メモリとプライベート・メモリが使われます。最低のメモリ量は，OpenVMS が必要とする量です。

表 10-1 ES47/ES80/GS1280 の構成

ES47	ES80	GS1280/8	GS1280/16	GS1280/32
2 インスタンス	4 インスタンス (組み込み IO を使用)	2 インスタンス	4 インスタンス (4 つの IO7 ポートが必要)	8 つの 4P インスタンス (8 つの IO7 ポートが必要)
2 つの 2P SBB	4 つの 2P SBB	1 つの 8P SBB	2 つの 8P SBB	4 つの 8P SBB
4 CPU	8 CPU	8 CPU	16 CPU	32 CPU

10.2 ステップ 2: ハードウェアの設定

構成に必要なハードウェアを取得したら，ハードウェア・マニュアルに記載された手順を適宜参照して取り付けてください。

10.3 ステップ 3: システム・ディスクの作成

次のどちらにするかを決めます。

- インスタンスごとにシステム・ディスクを 1 つ使用
- クラスタで共通のディスクを使用

10.4 ステップ 4: OpenVMS Alpha のインストール

OpenVMS Galaxy ソフトウェアを実行するのに，特別なインストール手順は必要ありません。Galaxy 機能は基本オペレーティング・システムに組み込まれており，この章で後述するコンソール・コマンドとシステム・パラメータ値を使用して，有効または無効に設定できます。

OpenVMS Alpha オペレーティング・システムのインストールの詳細については，『OpenVMS インストレーション・ガイド [翻訳版]』と，必要なパッチを参照してください。

ファームウェア・パッチの最新のリビジョン番号とファームウェア・パッチの入手方法については，次の Web ページで確認してください。

http://h18003.www1.hp.com/alphaserver/gs1280/gs1280_tech.html

10.4.1 OpenVMS Galaxy ライセンス情報

Galaxy 環境では，システム・スタートアップの際，およびインスタンス間の CPU 再割り当てが発生するたびに，OPENVMS-GALAXY ライセンス・ユニットがチェックされます。

CPU 起動時に CPU をサポートする OPENVMS-GALAXY ライセンス・ユニットが不足した場合，CPU はインスタンスの構成セットには存在しますが，停止されます。 後で適切なライセンス・ユニットをロードすると，停止している CPU をシステムの実行中に開始できます。

ライセンス管理についての詳細は，本書の付録C「ライセンスのインストール」と『OpenVMS License Management Utility Manual』を参照してください。

10.5 Step 5: パーティションの設定

オペレーティング・システムのインストールを終えると，ここに示す例のようにソフト・パーティションの設定を行うことができます。

AlphaServer GS1280 におけるサブパーティションを使った 32P ハード・パーティションの設定

この例では，32P GS1280 上の単一のハード・パーティション内に 3 つのソフト・パーティションを作成します。 1 つ目のパーティションは最初の 2 つの 8P ドローワで構成されます。 2 つ目と 3 つ目のパーティションは，それぞれ 1 つの 8P ドローワで構成されます。 共用メモリは 256 MB です。 show partition の表示でわかるように，各パーティションのコンソールは telnet ポートを通してアクセスできます。 power-on コマンドの後では show partition の表示が変わることに注意してください。

注意:

assign memory コマンドは，共用メモリの設定にのみ使用してください。

パート A: Galaxy の設定

```
Welcome - GS1280 Server Manager - V2.1-8
MBM> create partition -hp hp0 255 soft
MBM> create partition -hp hp0 -sp sp0
MBM> create partition -hp hp0 -sp sp1
MBM> create partition -hp hp0 -sp sp2
MBM> assign component -cab 0 -drawer 0 sbb -hp hp0 -sp sp0
MBM> assign component -cab 0 -drawer 1 sbb -hp hp0 -sp sp0
MBM> assign component -cab 0 -drawer 2 sbb -hp hp0 -sp sp1
MBM> assign component -cab 0 -drawer 3 sbb -hp hp0 -sp sp2
MBM> assign mem -hp hp0 256mb -com
MBM> show partition
------------------------------------------------------------------
Hard Partition : HP Name = hp0, HP No.= 0, SP count = 4
Attributes     : max CPUs = 255, SP type = soft, Non-stripe
Physical Memory: 69632MB (68.000GB)

 Community Memory: 256MB (0.250GB)

 Sub Partition: HP Name = hp0, HP No.= 0
                SP Name = sp0, SP No.= 0
                 State = Not Running, Telnet port = 323

 Assigned Memory: unspecified

 CPUs:
     Cab   Drw   CPU    (NS,EW)    PID     Type
      0     0     0     ( 0,0 )      0     Non-primary
      0     0     2     ( 0,1 )      2     Non-primary
      0     0     4     ( 0,2 )      4     Non-primary
```

```
      0    0    6    ( 0,3 )     6    Non-primary
      0    0    1    ( 1,0 )     1    Non-primary
      0    0    3    ( 1,1 )     3    Non-primary
      0    0    5    ( 1,2 )     5    Non-primary
      0    0    7    ( 1,3 )     7    Non-primary
      0    1    0    ( 2,0 )     8    Non-primary
      0    1    2    ( 2,1 )    10    Non-primary
      0    1    4    ( 2,2 )    12    Non-primary
      0    1    6    ( 2,3 )    14    Non-primary
      0    1    1    ( 3,0 )     9    Non-primary
      0    1    3    ( 3,1 )    11    Non-primary
      0    1    5    ( 3,2 )    13    Non-primary
      0    1    7    ( 3,3 )    15    Non-primary

IOPs:
           SBB                              PCI Drawer
   Cab  Drw  IOP   (NS,EW)            Cab  Drw  IOR
    0    0    0    ( 0,0 ) ---------  2    0    0
    0    0    2    ( 0,1 )
    0    0    4    ( 0,2 )
    0    0    6    ( 0,3 )
    0    0    1    ( 1,0 )
    0    0    3    ( 1,1 )
    0    0    5    ( 1,2 ) ---------  1    4    0
    0    0    7    ( 1,3 )
    0    1    0    ( 2,0 ) ---------  2    1    0
    0    1    2    ( 2,1 )
    0    1    4    ( 2,2 )
    0    1    6    ( 2,3 )
    0    1    1    ( 3,0 )
    0    1    3    ( 3,1 )
    0    1    5    ( 3,2 )
    0    1    7    ( 3,3 )

Sub Partition: HP Name = hp0, HP No.= 0
               SP Name = sp1, SP No.= 1
                State = Not Running, Telnet port = 324

Assigned Memory: unspecified

CPUs:
    Cab  Drw  CPU   (NS,EW)   PID    Type
     0    2    0    ( 0,4 )    0    Non-primary
     0    2    2    ( 0,5 )    2    Non-primary
     0    2    4    ( 0,6 )    4    Non-primary
     0    2    6    ( 0,7 )    6    Non-primary
     0    2    1    ( 1,4 )    1    Non-primary
     0    2    3    ( 1,5 )    3    Non-primary
     0    2    5    ( 1,6 )    5    Non-primary
     0    2    7    ( 1,7 )    7    Non-primary
IOPs:
           SBB                              PCI Drawer
    Cab  Drw  IOP   (NS,EW)            Cab  Drw  IOR
     0    2    0    ( 0,4 ) ---------  2    2    0
     0    2    2    ( 0,5 )
     0    2    4    ( 0,6 )
     0    2    6    ( 0,7 )
     0    2    1    ( 1,4 )
     0    2    3    ( 1,5 )
     0    2    5    ( 1,6 )
     0    2    7    ( 1,7 )

Sub Partition: HP Name = hp0, HP No.= 0
               SP Name = sp2, SP No.= 2
```

```
                State = Not Running, Telnet port = 325

 Assigned Memory: unspecified

 CPUs:
     Cab  Drw  CPU   (NS,EW)   PID    Type
      0    3    0    ( 2,4 )    8    Non-primary
      0    3    2    ( 2,5 )   10    Non-primary
      0    3    4    ( 2,6 )   12    Non-primary
      0    3    6    ( 2,7 )   14    Non-primary
      0    3    1    ( 3,4 )    9    Non-primary
      0    3    3    ( 3,5 )   11    Non-primary
      0    3    5    ( 3,6 )   13    Non-primary
      0    3    7    ( 3,7 )   15    Non-primary

 IOPs:
           SBB                            PCI Drawer
     Cab  Drw  IOP   (NS,EW)           Cab  Drw  IOR
      0    3    0    ( 2,4 ) ---------  2    3    0
      0    3    2    ( 2,5 )
      0    3    4    ( 2,6 )
      0    3    6    ( 2,7 )
      0    3    1    ( 3,4 )
      0    3    3    ( 3,5 )
      0    3    5    ( 3,6 )
      0    3    7    ( 3,7 )
Sub Partition: HP Name = hp0, HP No.= 0
               SP Name = Free_Pool, SP No.= 255
               State = Not Running
 Free Memory: 0MB (0.000GB)
 CPUs: None
 IOPs: None
MBM> power on -all
 [...]
```

パート B: インスタンス 0 からの出力:

```
~PCO-I-(pco_01) Powering on partition. HP: hp0
starting console on CPU 0
initialized idle PCB
initializing semaphores
initializing heap
initial heap 700c0
memory low limit = 54c000 heap = 700c0, 1fffc0
initializing driver structures
initializing idle process PID
initializing file system
initializing timer data structures
lowering IPL
CPU 0 speed is 1150 MHz
create dead_eater
create poll
create timer
create powerup
entering idle loop
access NVRAM
Get Partition DB
hpcount = 1, spcount = 4, ev7_count = 32, io7_count = 5
hard_partition = 0IO7-100 (Pass 3) at PID 8
IO7 North port speed is 191 MHz
Hose 32 - 33 MHz PCI
Hose 33 - 66 MHz PCI
Hose 34 - 66 MHz PCI
Hose 35 - 4X AGP
IO7-100 (Pass 3) at PID 0
IO7 North port speed is 191 MHz
Hose 0 - 33 MHz PCI
```

```
Hose 1 - 66 MHz PCI
Hose 2 - 66 MHz PCI
Hose 3 - 4X AGP
IO7-100 (Pass 3) at PID 5
IO7 North port speed is 191 MHz
Hose 20 - 33 MHz PCI
Hose 21 - 33 MHz PCI
Hose 22 - 33 MHz PCI
Hose 23 - 4X AGP
0 sub-partition 0:   start:00000000 00000000   size:00000000 80000000
PID 0 console memory base: 0, 2 GB
1 sub-partition 0:   start:00000004 00000000   size:00000000 80000000
PID 1 memory: 400000000, 2 GB
2 sub-partition 0:   start:00000008 00000000   size:00000000 80000000
PID 2 memory: 800000000, 2 GB
3 sub-partition 0:   start:0000000c 00000000   size:00000000 80000000
PID 3 memory: c00000000, 2 GB
4 sub-partition 0:   start:00000020 00000000   size:00000000 80000000
PID 4 memory: 2000000000, 2 GB
5 sub-partition 0:   start:00000024 00000000   size:00000000 80000000
PID 5 memory: 2400000000, 2 GB
6 sub-partition 0:   start:00000028 00000000   size:00000000 80000000
PID 6 memory: 2800000000, 2 GB
7 sub-partition 0:   start:0000002c 00000000   size:00000000 80000000
PID 7 memory: 2c00000000, 2 GB
8 sub-partition 0:   start:00000040 00000000   size:00000000 80000000
PID 8 memory: 4000000000, 2 GB
9 sub-partition 0:   start:00000044 00000000   size:00000000 80000000
PID 9 memory: 4400000000, 2 GB
10 sub-partition 0:   start:00000048 00000000   size:00000000 80000000
PID 10 memory: 4800000000, 2 GB
11 sub-partition 0:   start:0000004c 00000000   size:00000000 80000000
PID 11 memory: 4c00000000, 2 GB
12 sub-partition 0:   start:00000060 00000000   size:00000000 80000000
PID 12 memory: 6000000000, 2 GB
13 sub-partition 0:   start:00000064 00000000   size:00000000 80000000
PID 13 memory: 6400000000, 2 GB
14 sub-partition 0:   start:00000068 00000000   size:00000000 80000000
PID 14 memory: 6800000000, 2 GB
15 sub-partition 0:   start:0000006c 00000000   size:00000000 78000000
PID 15 memory: 6c00000000, 1.875 GB
0 sub-partition 1:   start:00000080 00000000   size:00000000 80000000
1 sub-partition 1:   start:00000084 00000000   size:00000000 80000000
2 sub-partition 1:   start:00000088 00000000   size:00000000 80000000
3 sub-partition 1:   start:0000008c 00000000   size:00000000 80000000
4 sub-partition 1:   start:000000a0 00000000   size:00000000 80000000
5 sub-partition 1:   start:000000a4 00000000   size:00000000 80000000
6 sub-partition 1:   start:000000a8 00000000   size:00000000 80000000
7 sub-partition 1:   start:000000ac 00000000   size:00000000 7c000000
0 sub-partition 2:   start:000000c0 00000000   size:00000000 80000000
1 sub-partition 2:   start:000000c4 00000000   size:00000000 80000000
2 sub-partition 2:   start:000000c8 00000000   size:00000001 00000000
3 sub-partition 2:   start:000000cc 00000000   size:00000001 00000000
4 sub-partition 2:   start:000000e0 00000000   size:00000000 80000000
5 sub-partition 2:   start:000000e4 00000000   size:00000000 80000000
6 sub-partition 2:   start:000000e8 00000000   size:00000000 80000000
7 sub-partition 2:   start:000000ec 00000000   size:00000000 7c000000
0 community 0:   start:0000006c 78000000   size:00000000 08000000
1 community 0:   start:000000ac 7c000000   size:00000000 04000000
2 community 0:   start:000000ec 7c000000   size:00000000 04000000
total memory, 31.875 GB
probe I/O subsystem
probing hose 0, PCI
probing PCI-to-PCI bridge, hose 0 bus 2
do not use secondary IDE channel on CMD controller
bus 2, slot 0, function 0 -- usba -- USB
bus 2, slot 0, function 1 -- usbb -- USB
bus 2, slot 0, function 2 -- usbc -- USB
bus 2, slot 0, function 3 -- usbd -- USB
bus 2, slot 1 -- dqa -- CMD 649 PCI-IDE
bus 2, slot 2 -- pka -- Adaptec AIC-7892
probing hose 1, PCI
```

```
bus 0, slot 1, function 0 -- pkb -- Adaptec AIC-7899
bus 0, slot 1, function 1 -- pkc -- Adaptec AIC-7899
probing hose 2, PCI
probing PCI-to-PCI bridge, hose 2 bus 2
bus 2, slot 4 -- eia -- DE602-B*
bus 2, slot 5 -- eib -- DE602-B*
bus 0, slot 2 -- pga -- FCA-2354
probing hose 3, PCI
probing hose 20, PCI
probing PCI-to-PCI bridge, hose 20 bus 2
do not use secondary IDE channel on CMD controller
bus 2, slot 0, function 0 -- usbe -- USB
bus 2, slot 0, function 1 -- usbf -- USB
bus 2, slot 0, function 2 -- usbg -- USB
bus 2, slot 0, function 3 -- usbh -- USB
bus 2, slot 1 -- dqb -- CMD 649 PCI-IDE
bus 2, slot 2 -- pkd -- Adaptec AIC-7892
probing hose 21, PCI
bus 0, slot 2 -- pgb -- KGPSA-C
probing hose 22, PCI
probing PCI-to-PCI bridge, hose 22 bus 2
bus 2, slot 4 -- eic -- DE602-AA
bus 2, slot 5 -- eid -- DE602-AA
probing hose 23, PCI
probing hose 32, PCI
probing PCI-to-PCI bridge, hose 32 bus 2
do not use secondary IDE channel on CMD controller
bus 2, slot 0, function 0 -- usbi -- USB
bus 2, slot 0, function 1 -- usbj -- USB
bus 2, slot 0, function 2 -- usbk -- USB
bus 2, slot 0, function 3 -- usbl -- USB
bus 2, slot 1 -- dqc -- CMD 649 PCI-IDE
bus 2, slot 2 -- pke -- Adaptec AIC-7892
probing hose 33, PCI
bus 0, slot 1, function 0 -- pkf -- Adaptec AIC-7899
bus 0, slot 1, function 1 -- pkg -- Adaptec AIC-7899
probing hose 34, PCI
probing PCI-to-PCI bridge, hose 34 bus 2
bus 2, slot 4 -- eie -- DE602-B*
bus 2, slot 5 -- eif -- DE602-B*
bus 0, slot 2 -- pgc -- FCA-2354
probing hose 35, PCI
starting drivers
Starting secondary CPU 1 at address 400030000
Starting secondary CPU 2 at address 800030000
Starting secondary CPU 3 at address c00030000
Starting secondary CPU 4 at address 2000030000
Starting secondary CPU 5 at address 2400030000
Starting secondary CPU 6 at address 2800030000
Starting secondary CPU 7 at address 2c00030000
Starting secondary CPU 8 at address 4000030000
Starting secondary CPU 9 at address 4400030000
Starting secondary CPU 10 at address 4800030000
Starting secondary CPU 11 at address 4c00030000
Starting secondary CPU 12 at address 6000030000
Starting secondary CPU 13 at address 6400030000
Starting secondary CPU 14 at address 6800030000
Starting secondary CPU 15 at address 6c00030000
initializing GCT/FRU.....................................................
..................... at 54c000
Initializing dqa dqb dqc eia eib eic eid eie eif pka pkb pkc pkd pke pkf
 pkg pga pgb  pgc
AlphaServer Console T6.5-14, built on Jun 20 2003 at 14:52:48
P00>>>
```

各パーティション用に個別に環境変数を設定します。

```
P00>>> set ei*mode twi
P00>>> set bootdef_dev dka0
```

パート C: インスタンス 1 からの出力:

```
~PCO-I-(pco_01) Powering on partition. HP: hp0
starting console on CPU 16
console physical memory base is 8000000000
initialized idle PCB
initializing semaphores
initializing heap
initial heap 700c0
memory low limit = 54c000 heap = 700c0, 1fffc0
initializing driver structures
initializing idle process PID
initializing file system
initializing timer data structures
lowering IPL
CPU 16 speed is 1150 MHz
create dead_eater
create poll
create timer
create powerup
entering idle loop
access NVRAM
Get Partition DB
hpcount = 1, spcount = 4, ev7_count = 32, io7_count = 5
hard_partition = 0
IO7-100 (Pass 3) at PID 10
IO7 North port speed is 191 MHz
Hose 64 - 33 MHz PCI

Hose 65 - 66 MHz PCI
Hose 66 - 66 MHz PCI
Hose 67 - 4X AGP
0 sub-partition 0:   start:00000000 00000000   size:00000000 80000000
1 sub-partition 0:   start:00000004 00000000   size:00000000 80000000
2 sub-partition 0:   start:00000008 00000000   size:00000000 80000000
3 sub-partition 0:   start:0000000c 00000000   size:00000000 80000000
4 sub-partition 0:   start:00000020 00000000   size:00000000 80000000
5 sub-partition 0:   start:00000024 00000000   size:00000000 80000000
6 sub-partition 0:   start:00000028 00000000   size:00000000 80000000
7 sub-partition 0:   start:0000002c 00000000   size:00000000 80000000
8 sub-partition 0:   start:00000040 00000000   size:00000000 80000000
9 sub-partition 0:   start:00000044 00000000   size:00000000 80000000
10 sub-partition 0:   start:00000048 00000000   size:00000000 80000000
11 sub-partition 0:   start:0000004c 00000000   size:00000000 80000000
12 sub-partition 0:   start:00000060 00000000   size:00000000 80000000
13 sub-partition 0:   start:00000064 00000000   size:00000000 80000000
14 sub-partition 0:   start:00000068 00000000   size:00000000 80000000
15 sub-partition 0:   start:0000006c 00000000   size:00000000 78000000
0 sub-partition 1:   start:00000080 00000000   size:00000000 80000000
PID 16 console memory base: 8000000000, 2 GB
1 sub-partition 1:   start:00000084 00000000   size:00000000 80000000
PID 17 memory: 8400000000, 2 GB
2 sub-partition 1:   start:00000088 00000000   size:00000000 80000000
PID 18 memory: 8800000000, 2 GB
3 sub-partition 1:   start:0000008c 00000000   size:00000000 80000000
PID 19 memory: 8c00000000, 2 GB
4 sub-partition 1:   start:000000a0 00000000   size:00000000 80000000
PID 20 memory: a000000000, 2 GB
5 sub-partition 1:   start:000000a4 00000000   size:00000000 80000000
PID 21 memory: a400000000, 2 GB
6 sub-partition 1:   start:000000a8 00000000   size:00000000 80000000
PID 22 memory: a800000000, 2 GB
7 sub-partition 1:   start:000000ac 00000000   size:00000000 7c000000
PID 23 memory: ac00000000, 1.938 GB
0 sub-partition 2:   start:000000c0 00000000   size:00000000 80000000
```

```
1 sub-partition 2:   start:000000c4 00000000   size:00000000 80000000
2 sub-partition 2:   start:000000c8 00000000   size:00000001 00000000
3 sub-partition 2:   start:000000cc 00000000   size:00000001 00000000
4 sub-partition 2:   start:000000e0 00000000   size:00000000 80000000
5 sub-partition 2:   start:000000e4 00000000   size:00000000 80000000
6 sub-partition 2:   start:000000e8 00000000   size:00000000 80000000
7 sub-partition 2:   start:000000ec 00000000   size:00000000 7c000000
0 community 0:   start:0000006c 78000000   size:00000000 08000000
1 community 0:   start:000000ac 7c000000   size:00000000 04000000
2 community 0:   start:000000ec 7c000000   size:00000000 04000000
total memory, 15.938 GB
waiting for GCT/FRU at 54c000 by CPU 0
probe I/O subsystem
probing hose 64, PCI
probing PCI-to-PCI bridge, hose 64 bus 2
do not use secondary IDE channel on CMD controller
bus 2, slot 0, function 0 -- usba -- USB
bus 2, slot 0, function 1 -- usbb -- USB
bus 2, slot 0, function 2 -- usbc -- USB
bus 2, slot 0, function 3 -- usbd -- USB
bus 2, slot 1 -- dqa -- CMD 649 PCI-IDE
bus 2, slot 2 -- pka -- Adaptec AIC-7892
probing hose 65, PCI
bus 0, slot 1, function 0 -- pkb -- Adaptec AIC-7899
bus 0, slot 1, function 1 -- pkc -- Adaptec AIC-7899
probing hose 66, PCIprobing PCI-to-PCI bridge, hose 66 bus 2
bus 2, slot 4 -- eia -- DE602-B*
bus 2, slot 5 -- eib -- DE602-B*
bus 0, slot 2 -- pga -- FCA-2354
probing hose 67, PCI
starting drivers
Starting secondary CPU 17 at address 8400030000
Starting secondary CPU 18 at address 8800030000
Starting secondary CPU 19 at address 8c00030000
Starting secondary CPU 20 at address a000030000
Starting secondary CPU 21 at address a400030000
Starting secondary CPU 22 at address a800030000
Starting secondary CPU 23 at address ac00030000
initializing GCT/FRU to 54c000
Initializing dqa eia eib pka pkb pkc pga
AlphaServer Console T6.5-14, built on Jun 20 2003 at 14:52:48
P16>>> set ei*mode twi
P16>>> set bootdef_dev dka0
```

パート D: インスタンス 2 からの出力:

```
~PCO-I-(pco_01) Powering on partition. HP: hp0
starting console on CPU 24
console physical memory base is c000000000
initialized idle PCB
initializing semaphores
initializing heap
initial heap 700c0
memory low limit = 54c000 heap = 700c0, 1fffc0
initializing driver structures
initializing idle process PID
initializing file system
initializing timer data structures
lowering IPL
CPU 24 speed is 1150 MHz
create dead_eater
create poll
create timer
create powerup
entering idle loop
```

```
access NVRAM
Get Partition DB
hpcount = 1, spcount = 4, ev7_count = 32, io7_count = 5
hard_partition = 0
IO7-100 (Pass 3) at PID 18
IO7 North port speed is 191 MHz
Hose 96 - 33 MHz PCI
Hose 97 - 66 MHz PCI
Hose 98 - 66 MHz PCI
Hose 99 - 4X AGP
0 sub-partition 0:   start:00000000 00000000   size:00000000 80000000
1 sub-partition 0:   start:00000004 00000000   size:00000000 80000000
2 sub-partition 0:   start:00000008 00000000   size:00000000 80000000
3 sub-partition 0:   start:0000000c 00000000   size:00000000 80000000
4 sub-partition 0:   start:00000020 00000000   size:00000000 80000000
5 sub-partition 0:   start:00000024 00000000   size:00000000 80000000
6 sub-partition 0:   start:00000028 00000000   size:00000000 80000000
7 sub-partition 0:   start:0000002c 00000000   size:00000000 80000000
8 sub-partition 0:   start:00000040 00000000   size:00000000 80000000
9 sub-partition 0:   start:00000044 00000000   size:00000000 80000000
10 sub-partition 0:   start:00000048 00000000   size:00000000 80000000
11 sub-partition 0:   start:0000004c 00000000   size:00000000 80000000
12 sub-partition 0:   start:00000060 00000000   size:00000000 80000000
13 sub-partition 0:   start:00000064 00000000   size:00000000 80000000
14 sub-partition 0:   start:00000068 00000000   size:00000000 80000000
15 sub-partition 0:   start:0000006c 00000000   size:00000000 78000000
0 sub-partition 1:   start:00000080 00000000   size:00000000 80000000
1 sub-partition 1:   start:00000084 00000000   size:00000000 80000000
2 sub-partition 1:   start:00000088 00000000   size:00000000 80000000
3 sub-partition 1:   start:0000008c 00000000   size:00000000 80000000
4 sub-partition 1:   start:000000a0 00000000   size:00000000 80000000
5 sub-partition 1:   start:000000a4 00000000   size:00000000 80000000
6 sub-partition 1:   start:000000a8 00000000   size:00000000 80000000
7 sub-partition 1:   start:000000ac 00000000   size:00000000 7c000000
0 sub-partition 2:   start:000000c0 00000000   size:00000000 80000000
PID 24 console memory base: c000000000, 2 GB
1 sub-partition 2:   start:000000c4 00000000   size:00000000 80000000
PID 25 memory: c400000000, 2 GB
2 sub-partition 2:   start:000000c8 00000000   size:00000001 00000000
PID 26 memory: c800000000, 4 GB
3 sub-partition 2:   start:000000cc 00000000   size:00000001 00000000
PID 27 memory: cc00000000, 4 GB
4 sub-partition 2:   start:000000e0 00000000   size:00000000 80000000
PID 28 memory: e000000000, 2 GB
5 sub-partition 2:   start:000000e4 00000000   size:00000000 80000000
PID 29 memory: e400000000, 2 GB
6 sub-partition 2:   start:000000e8 00000000   size:00000000 80000000
PID 30 memory: e800000000, 2 GB
7 sub-partition 2:   start:000000ec 00000000   size:00000000 7c000000
PID 31 memory: ec00000000, 1.938 GB
0 community 0:   start:0000006c 78000000   size:00000000 08000000
1 community 0:   start:000000ac 7c000000   size:00000000 04000000
2 community 0:   start:000000ec 7c000000   size:00000000 04000000
total memory, 19.938 GB
waiting for GCT/FRU at 54c000 by CPU 0
waiting for GCT/FRU at 54c000 by CPU 0
waiting for GCT/FRU at 54c000 by CPU 0
waiting for GCT/FRU at 54c000 by CPU 0
waiting for GCT/FRU at 54c000 by CPU 0
waiting for GCT/FRU at 54c000 by CPU 0
waiting for GCT/FRU at 54c000 by CPU 0
waiting for GCT/FRU at 54c000 by CPU 0
waiting for GCT/FRU at 54c000 by CPU 0
waiting for GCT/FRU at 54c000 by CPU 0
waiting for GCT/FRU at 54c000 by CPU 0
```

```
waiting for GCT/FRU at 54c000 by CPU 0
waiting for GCT/FRU at 54c000 by CPU 0
waiting for GCT/FRU at 54c000 by CPU 0
waiting for GCT/FRU at 54c000 by CPU 0
waiting for GCT/FRU at 54c000 by CPU 0
waiting for GCT/FRU at 54c000 by CPU 0
waiting for GCT/FRU at 54c000 by CPU 0
waiting for GCT/FRU at 54c000 by CPU 0
probe I/O subsystem
probing hose 96, PCI
probing PCI-to-PCI bridge, hose 96 bus 2
do not use secondary IDE channel on CMD controller
bus 2, slot 0, function 0 -- usba -- USB
bus 2, slot 0, function 1 -- usbb -- USB
bus 2, slot 0, function 2 -- usbc -- USB
bus 2, slot 0, function 3 -- usbd -- USB
bus 2, slot 1 -- dqa -- CMD 649 PCI-IDE
bus 2, slot 2 -- pka -- Adaptec AIC-7892
probing hose 97, PCI
bus 0, slot 1, function 0 -- pkb -- Adaptec AIC-7899
bus 0, slot 1, function 1 -- pkc -- Adaptec AIC-7899
probing hose 98, PCI
probing PCI-to-PCI bridge, hose 98 bus 2
bus 2, slot 4 -- eia -- DE602-B*
bus 2, slot 5 -- eib -- DE602-B*
bus 0, slot 2 -- pga -- FCA-2354
probing hose 99, PCI
starting drivers
Starting secondary CPU 25 at address c400030000
Starting secondary CPU 26 at address c800030000
Starting secondary CPU 27 at address cc00030000
Starting secondary CPU 28 at address e000030000
Starting secondary CPU 29 at address e400030000
Starting secondary CPU 30 at address e800030000
Starting secondary CPU 31 at address ec00030000
initializing GCT/FRU to 54c000
Initializing dqa eia eib pka pkb pkc pga
AlphaServer Console T6.5-14, built on Jun 20 2003 at 14:52:48
P24>>> set ei*mode two
P24>>> set bootdef_dev dka100
```

10.6 ステップ 6: OpenVMS Galaxy のブート

パーティションの設定を変更すると，コンソール環境変数の値に影響を与えることがあります。 環境変数の値が正しいことを確認してください。 初期インストールの後，またシステムの障害やオペレータによって要求されたリブートの後，システムをリブートする必要がある場合は，AUTOGEN が正しくリブートされるように，ブートの前に各 Galaxy コンソールで BOOTDEF_DEV 変数と BOOT_OSFLAGS 変数を設定してください。

Galaxy を有効にして OpenVMS システムをブートすると，各ソフト・パーティションでは，オペレーティング・システムの個別のインスタンスを Galaxy のメンバとして実行できます。

ブート・プロセスのある時点で，インスタンスは Galaxy 構成に参加しようとします。 その際，Galaxy が存在しないと，Galaxy が作成され，新しい Galaxy ID が生成され，共用メモリ内のデータ構造が初期化されます。 このデータ構造は，その Galaxy のすべてのメンバが使うことになります。 Galaxy が存在すると，そのインスタンスは Galaxy 構成に参加し，共用メモリ構造に接続し，共用メモリを通じて他のメンバ・インスタンスに接続します。

ES47/ES80/GS1280 上の Galaxy のインスタンスでは，次の制限があります。

- 各 2P ビルディング・ブロック内には，プライマリ CPU は 1 つしか存在できない。
- 2P ビルディング・ブロックの CPU のうちのどちらかが，Galaxy インスタンス内でプライマリになることができる。
- 各 Galaxy には IO ポートが必要。

- ハード・パーティションあたり，最大 8 つの Galaxy パーティションがサポートされる。

Galaxy ファームウェアを正しくインストールし，コンソールを構成した後，次のようにして初期 Galaxy 環境をブートできます。

各 Galaxy インスタンスに対して，次の操作を実行します。

```
P00>>> B -FL 0,1 DKA100 // or whatever your boot device is.

SYSBOOT> SET GALAXY 1

SYSBOOT> CONTINUE
```

構成が完了しました。これで OpenVMS Galaxy が構築されました。

第11章　Alpha システムでのシングル・インスタンス Galaxy の使用

OpenVMS Alpha バージョン 7.2 以降，ES45 を除く Alpha プラットフォームでシングル・インスタンス Galaxy を実行できるようになりました。 この機能により，早い段階で OpenVMS Galaxy の機能を評価することができ， 最も重要なこととして，OpenVMS の複数のインスタンスを実行できるシステム (たとえば AlphaServer 8400) で完全なスケールの Galaxy コンピューティング環境を設定する前に， Galaxy 対応アプリケーションを開発し，テストすることができます。

Alpha システムで動作するシングル・インスタンス Galaxy はエミュレータではありません。これは Galaxy インタフェースと基礎になるオペレーティング・システム機能を備えた OpenVMS Galaxy コードです。 シングル・インスタンス Galaxy には，すべての Galaxy API が備わっています (たとえばリソース管理，共用メモリ・アクセス，イベント通知，同期のためのロック，グローバル・セクションのための共用メモリなど)。

シングル・インスタンス Galaxy で動作するアプリケーションは， マルチインスタンス Galaxy システムと同じオペレーティング・システム・コードを実行します。この機能は，構成ファイル SYS$SYSTEM:GLX$GCT.BIN を作成することで実現されます。 OpenVMS はこのファイルをメモリに読み込みます。Galaxy プラットフォーム (たとえば AlphaServer 8400) で， コンソールは使用する OpenVMS 用のメモリに構成データを格納します。 構成データがメモリに格納されると，それがどこからロードされたかとは無関係に， OpenVMS は Galaxy インスタンスとしてブートされます。

11.1 GCU を使ったシングル・インスタンス Galaxy の作成

Galaxy Configuration Utility (GCU) を使用して Alpha システムでシングル・インスタンス Galaxy を作成するには，次の手順を実行します。

1. シングル・インスタンス Galaxy を使用する OpenVMS Alpha システムで GCU を実行します。
2. GCU が Galaxy 以外のシステムで実行された場合は， シングル・インスタンス Galaxy を作成するかどうか質問されます。 **OK** をクリックします。
3. 次に GCU は共用メモリの容量を質問します。適切な値 (8MB の倍数) を入力します。Galaxy インスタンスとしてブートする場合は， 少なくとも 8MB の共用メモリを指定しなければなりません。
4. 一度 GCU が構成を表示したら，その時点ですでにファイル GLX$GCT.BIN がカレント・ディレクトリに書き込まれています。 この時点で GCU を終了できます。 操作ミスした場合や，構成を変更する場合は，現在のモデルを終了し， 処理を繰り返すことができます。

11.2 Galaxy インスタンスとしてのリブート

システムを Galaxy インスタンスとしてリブートするには，次のようにします。

1. GLX$GCT.BIN ファイルを SYS$SYSROOT:[SYSEXE]GLX$GCT.BIN にコピーします。
2. システムをシャットダウンします。
3. 会話型ブート・コマンドを使用してリブートします。次に例を示します。

```
>>> B -FL 0,1 device
```

4. 次のコマンドを入力します。

```
SYSBOOT> SET GALAXY 1
SYSBOOT> CONTINUE
```

5. GALAXY=1 を SYS$SYSTEM:MODPARAMS.DAT に追加します。

第12章　OpenVMS Galaxy に関するヒントと手法

この章には OpenVMS エンジニアリング・グループが OpenVMS Galaxy 環境の作成と実行で役に立つと見なした情報を記載します。

12.1 システム・オート・アクション

システムの電源投入時に，AUTO_ACTION コンソール環境変数がインスタンス 0 に対して BOOT または RESTART に設定されている場合， GALAXY コマンドが自動的に実行され，インスタンス 0 がブートしようとします。

他のインスタンスに対するコンソール環境変数の AUTO_ACTION の設定は， GALAXY コマンドの入力時にそれぞれの動作を行います (自動的に実行される場合も，ユーザがコンソールから実行する場合も)。

この機能を使用するようにシステムを設定するには， 各インスタンスでコンソール環境変数 AUTO_ACTION を RESTART または BOOT を設定しなければなりません。 BOOT_OSFLAGS 環境変数と BOOTDEF_DEV 環境変数に適切な値を指定しなければなりません。

12.2 コンソール環境変数の変更

OpenVMS Galaxy 操作用の LP_* 環境変数の初期セットを設定し， ソフト・パーティションを起動した後で環境変数値を変更するには， まずコンソール・システムを初期化してソフト・パーティションを停止し， 値を変更してから，もう一度コンソール・システムを初期化する必要があります。 新しい値をすべてのパーティションに適切に行き渡らせるには， 2 度の初期化のあいだに変更を終えておく必要があります。その後でソフト・パーティションの起動を行います。

注意:

AlphaServer 4100 システムでは起動に INIT コマンドは必要ありませんが， 両方のインスタンスでこれらの変数を変更しなければなりません。

12.3 コンソールに関するヒント

AlphaServer 8400 および 8200 システムは， Galaxy ソフトウェア・アーキテクチャが開発される以前に設計されているため，OpenVMS Galaxy コンソール・ファームウェアとシステム・オペレーションで， いくつかの制限事項に対応しなければなりません。

ここでは，認識しておかなければならない事項と回避しなければならない事項について説明します。

- BOOT_RESET 環境変数を 1 に設定しないでください。このように設定すると， 各セカンダリ・コンソールがブートの前にバスをリセットするので， すでにブートされているすべてのパーティションがリセットされます。OpenVMS Galaxy パーティションはハードウェアを共用することに注意してください。
- 処理が完了するまで我慢強く待ってください。コンソールの初期化とシステムのリブートには，数分かかることがあります。
- **ファームウェア更新プロセスを途中で打ち切らないでください。**

  途中で打ち切ると，使用しているシステムがハングします。
- コンソール・ファームウェアを更新する場合は， 同時に **すべての CPU** を更新してください。

  2 種類の異なる CPU や 2 種類の異なるファームウェア・リビジョンを実行しないでください。 一貫性のあるファームウェア・リビジョンを提供しないと， システムは電源投入時にハングします。

- セカンダリ・コンソールから **GALAXY** コマンドを絶対に入力しないでください。 この操作を実行すると，システムが再初期化されるので，プライマリ・コンソールで操作を最初からやり直さなければなりません。

12.4 Galaxy モードのオフ設定

OpenVMS Galaxy ソフトウェアをオフに設定する場合は， LP_COUNT 環境変数を次のように変更し，次のコマンドを入力します。

```
>>> SET LP_COUNT 0    ! Return to monolithic SMP config
>>> INIT              ! Return to single SMP console
>>> B -fl 0,1 device  ! Stop at SYSBOOT
SYSBOOT> SET GALAXY 0
SYSBOOT> CONTINUE
```

第13章　OpenVMS Galaxy Configuration ユーティリティ

Galaxy Configuration ユーティリティ (GCU) は DECwindows Motif アプリケーションであり，システム管理者は GCU を使用することで，1 つのワークステーション・ウィンドウから OpenVMS Galaxy システムを構成し，管理することができます。

GCU を使用すると，システム管理者は次の操作を実行できます。

- アクティブな Galaxy 構成を表示できます。
- Galaxy インスタンス間でリソースの再割り当てを実行できます。
- リソース固有の属性を表示できます。
- 1 つ以上の Galaxy インスタンスをシャットダウンまたはリブートできます。
- 追加管理ツールを起動できます。
- Galaxy 構成モデルを作成し，そのモデルに設定することができます。
- Alpha システムでシングル・インスタンス Galaxy を構築できます (Galaxy 以外のハードウェア・プラットフォームでソフトウェアを開発するため)。
- Galaxy に関するオンライン・ドキュメントを表示できます。
- 現在のハードウェア・プラットフォームのホット・スワップ属性を判断できます。

GCU は SYS$SYSTEM ディレクトリにあり，このディレクトリには構成に関する情報を格納したファイルもいくつかあります。

GCU は次のファイルで構成されます。

ファイル	説明
SYS$SYSTEM:GCU.EXE	GCU 実行可能イメージ
SYS$MANAGER:GCU.DAT	オプションの DECwindows リソース・ファイル
SYS$MANAGER:GALAXY.GCR	Galaxy 構成ルールセット
SYS$MANAGER:GCU$ACTIONS.COM	システム管理プロシージャ
SYS$MANAGER:xxx.GCM	ユーザ定義構成モデル
SYS$HELP:GALAXY_GUIDE.DECW$BOOK	Bookreader 形式のオンライン・ヘルプ

GCU はどの Galaxy インスタンスからも実行できます。システムでグラフィックス出力が直接サポートされない場合は，DECwindows ディスプレイを外部ワークステーションまたは適切な構成の PC に設定できます。しかし，GCU アプリケーション自体は，常に Galaxy システム上で動作していなければなりません。

GCU が起動されると，リソース・ファイル (GCU.DAT) から検出したカスタマイゼーションがロードされます。その後，Galaxy 構成ルールセット (GALAXY.GCR) がロードされます。ルールセット・ファイルには，GCU がさまざまなシステム・コンポーネントを表示する方法を指定する文 (statement) が格納されており，ユーザが構成表示と会話する方法を管理するルールも含まれています。ルールセット・ファイルの構造に精通しているか，または弊社のサービス・エンジニアから依頼された場合を除き，ユーザがルールセット・ファイルを変更することはありません。

GCU ディスプレイが表示可能になると，GCU はシステムが現在 OpenVMS Galaxy として構成されているのか，Galaxy 以外のプラットフォームでシングル・インスタンス Galaxy として構成されているのかを判断します。システムが Galaxy として構成されている場合は，GCU はアクティブ Galaxy 構成モデルを表示します。メイン・ウィンドウには Galaxy の構成が階層構造で表示されます。システムが Galaxy として構成されていない場合は，シングル・インスタンス Galaxy を作成するかどうかが質問されます。GCU はどの Alpha システムでもシングル・インスタンス Galaxy を作成できますが，マルチインスタンス OpenVMS Galaxy 環

境は，コンソール・コマンドとコンソール環境変数を使用して作成されることに注意してください。

Galaxy 構成モデルが表示された後， ユーザはアクティブ・モデルと会話することができ，モデルをオフラインにして， 後で使用するために特定の構成を定義することもできます。 この後の節では，これらの機能について詳しく説明します。

13.1 GCU の概略

GCU では次の 3 種類の操作を実行できます。

- Galaxy 構成モデルまたはシングル・インスタンス Galaxy を作成できます (13.1.1 項「Galaxy 構成モデルの作成」)。
- アクティブな Galaxy リソースを監視できます (13.1.2 項「監視」)。
- アクティブな Galaxy リソース構成とやり取りできます (13.1.3 項「会話」)。

ほとんどの GCU 操作は，メインの監視ウィンドウおよび Galaxy コンポーネントを表す階層構造を中心にして行われます。 監視ウィンドウは非常に大きな空間への入口として機能します。 監視ウィンドウは，必要に応じて左右上下に移動したり，拡大/縮小することができ，Galaxy 構成全体またはその一部を監視することができます。 メイン・ツールバーには， ワークスペース・ズーム操作を制御する複数のボタンが表示されます。ワークスペースのパン操作は， 水平スクロール・バーと垂直スクロール・バーで制御されます。 ワークスペースのスライドは， マウスの中央ボタンを押しながらワークスペースをドラッグすることで実行されます。 このためには 3 ボタン・マウスを使用する必要があります。

さまざまな GCU 操作はプルダウン・メニューやポップアップ・メニューから起動されます。ファイルを開いたり閉じる操作，外部ツールを起動する操作などの全般的な操作は，メイン・メニュー・バー・エントリから実行できます。 各 Galaxy コンポーネント固有の操作は，監視ウィンドウに表示されるコンポーネントの上でマウスの右ボタンをクリックしたときに表示されるポップアップ・メニューを使用して行うことができます。

多くの操作に対する応答として，GCU は追加ダイアログ・ボックスを表示します。 このダイアログ・ボックスには情報，フォーム，エディタ， プロンプトなどが表示されます。 エラーや情報メッセージは，エラーの重大度やメッセージの重要性に応じて，ポップアップ・ダイアログ・ボックスに表示されるか， ウィンドウの下部のステータス・バーに表示されます。

13.1.1 Galaxy 構成モデルの作成

GCU を使用すると，Galaxy 構成モデルを作成でき， Alpha システムでシングル・インスタンス Galaxy を作成することもできます。

アクティブ Galaxy 構成モデルを表示しているときに， 表示オブジェクト (コンポーネント) を直接操作すると， 動作中の構成が変更されることがあります。 たとえば，CPU を現在の位置からドラッグして，別のインスタンス・コンポーネントの上にドロップすると， 管理アクション・プロシージャが起動され， 選択した CPU が新しいインスタンスに再割り当てされます。場合によっては，このような操作が適切なこともありますが， コンポーネントごとに構成を変更するのではなく， Galaxy 全体を再構成しなければならないこともあります。 その場合は，オフラインの Galaxy 構成モデルを作成する必要があります。

Galaxy 構成モデルを作成するには， 既存のモデル (通常はアクティブなモデル) を使用して，そのモデルを必要に応じて変更し，ファイルに保存します。

アクティブな Galaxy 構成モデルの利用:

1. ENGAGE ボタンを押すと， モデルは DISENGAGED (設定が解除された状態) になります。ボタンは赤から白に変わり，押されない状態になるはずです。 DISENGAGED 状態になると，すべての CPU コンポーネントは赤になり， 割り当てられていない状態であることを示します。 CPU がシャットダウンされたわけではないので，驚かないでください。
2. 各 CPU を適切なインスタンスにドラッグ・アンド・ドロップして， CPU の割り当てを変更します。

3. 割り当ての変更が終了したら，そのモデルに再設定することができ，後で使用するためにモデルをファイルに保存することもできます。 モデルに再び設定すると，モデルがアクティブ・モデルから作成された場合も，ファイルに保存されているモデルの場合も，GCU はアクティブ・システム構成をモデルから提案された構成と比較します。その後，システムを新しいモデルに再割り当てするのに必要な管理アクションの概要を示します。ユーザがそれらのアクションを承認すると，GCU は必要な管理アクションを実行し，作成されたモデルは，設定されているアクティブなモデルとして表示されます。

オフライン・モデルを作成するのは，大幅な構成の変更を自動化するためです。たとえば，この手順に従えば，適切な Galaxy 構成を表すモデルを必要に応じて作成しておき，会話しながらそのモデルに設定することができます。

13.1.2 監視

GCU は 1 つのアクティブな Galaxy 構成モデルを表示することができ，複数のオフライン Galaxy 構成モデルを表示することもできます。 ロードされた各モデルは，ツールバーの [Model] メニューにアイテムとして表示されます。 適切なメニュー・アイテムをクリックすれば，モデルを切り換えることができます。

アクティブ・モデルは常に GLX$ACTIVE.GCM という名前です。 アクティブ・モデルが最初にロードされると，システムがシステム・ハードウェアをもとにモデルを確認する間，しばらくこの名前のファイルが存在します。

モデルが表示されると，必要に応じてズーム，パン，スライドによって，Galaxy コンポーネントを表示できます。ツールバーの左側のボタンを使用すれば，ズーム機能を制御できます。

次のズーム機能を利用できます。

機能	説明
Galactic zoom	コンポーネント階層構造全体が監視ウィンドウに表示されるようにズームする。
Zoom 1:1	コンポーネントを通常のスケールにズームする。
Zoom to region	選択された表示領域だけを表示するようにズームする。
Zoom in	10 パーセント拡大する。
Zoom out	10 パーセント縮小する。

パン操作は，垂直スクロール・バーと水平スクロール・バーを使用して行います。スライド操作はマウスの中央ボタンを押しながら，カーソルとイメージをドラッグ (スライド) させることで行います。

13.1.2.1 レイアウト管理

自動レイアウト機能はコンポーネントのレイアウトを管理します。自動レイアウト・モードでレイアウトの表示を更新しなければならない場合は，ルート (一番上) コンポーネントを選択します。

現在のレイアウトを変更するには，[Windows] メニューから [Manual Layout] を選択します。手動レイアウト・モードでは，コンポーネントをドラッグ・アンド・ドロップすることができますが，わかりやすい構造を作成するようにしてください。 各コンポーネントは自動的なレイアウト制限に拘束されないため，時間をかけて各チャートで各コンポーネントの配置を考慮する必要があります。 簡単に操作できるようにするために，マウスの右ボタンを任意のコンポーネントの上でクリックして，レイアウト・サブツリーを選択すれば，階層構造のその場所より下の部分で，自動的なレイアウト・アシスタンス機能が提供されます。

満足できるレイアウトが作成されたら，現在のモデルをファイルに保存して，手動レイアウト情報を残しておかなければなりません。カスタム・レイアウトは，モデルが開かれるときに使用されます。自動レイアウト・モードを選択すると，手動レイアウトはメモリ内のモデルから失われます。 また，CPU コンポーネントを表示上効果的な方法で再割り当てするには，インスタンス・レベルの下でサブツリー・レイアウト操作を実行しなければなりません。この理由

から，手動レイアウト操作は，コンポーネント階層構造のインスタンス・レベルおよびコミュニティ・レベルに制限するのが適切です。

13.1.2.2 OpenVMS Galaxy チャート

GCU では，チャートと呼ぶ 6 種類のモデルのサブセットが提供されます。 6 種類のチャートは次のとおりです。

チャート名	表示の内容
Logical Structure	動的リソース割り当て
Physical Structure	変化しないハードウェアの関係
CPU Assignment	CPU 割り当ての単純な表示
Memory Assignment	メモリ・サブシステム・コンポーネント
IOP Assignment	I/O モジュールの関係
Failover Targets	プロセッサ・フェールオーバ割り当て

これらのチャートは，さまざまな種類のコンポーネントの表示を有効または無効に設定して，関係するコンポーネントのサブセットの表示を提供することで作成されます。

各チャートには，そのチャート固有の機能があります。 たとえば，CPU の再割り当てでは，インスタンス・コンポーネントが表示されていなければなりません。 インスタンスは Physical Structure チャートや Memory Assignment チャートに表示されないので，CPU の再割り当ては Logical Structure チャートおよび CPU Assignment チャートでのみ実行できます。

チャートの詳細については，13.4 項「GCU チャートの使用」 を参照してください。

13.1.3 会話

アクティブ Galaxy 構成モデルを表示している間， システム・コンポーネントと直接会話することができます。 たとえば，CPU をあるインスタンスから別のインスタンスに再割り当てするには， CPU を適切なインスタンスにドラッグ・アンド・ドロップします。 GCU はその操作が有効であるかどうか確認し， 外部コマンド・アクションを実行して構成を変更します。 設定されていないモデルとの会話は， 単にオフライン・モデルに対する描画操作であり， 稼働中のシステムには影響しません。

Galaxy コンポーネントと会話している間，GCU は誤った構成や不適切な管理アクションが実行されないように， 組み込みルールとユーザ定義ルールを適用します。 たとえば，プライマリ CPU を再割り当てすることはできず， CPU を Galaxy インスタンス以外のコンポーネントに再割り当てすることもできません。 このような操作を実行すると，ステータス・バーにエラー・メッセージが表示され， モデルは適切な構成に戻ります。 実行しようとした操作が構成ルールのいずれかに違反する場合は，ステータス・バーに赤で表示されるエラー・メッセージに， 違反するルールが示されます。

マウスの右ボタンをクリックし， ポップアップ・メニューから [Parameters] を選択するか，メイン・ツールバーの [Components] メニューから [Parameters] を選択すると， 選択したコンポーネントの詳細情報を表示できます。

[Galaxy] メニューの [Shutdown] または [Reboot] を使用すると， GCU は 1 つ以上の Galaxy インスタンスをシャットダウンまたはリブートすることができます。 さまざまなシャットダウン・パラメータやリブート・パラメータは，[Shutdown] ダイアログ・ボックスに入力できます。 クラスタ接続された Galaxy インスタンスを完全にシャットダウンするには，CLUSTER_SHUTDOWN オプションを指定する必要があります。 [Shutdown] ダイアログ・ボックスでは，インスタンスの組み合わせまたはすべてのインスタンスを選択できます。 GCU は非常に賢明なので，所有者のインスタンスを最後にシャットダウンします。

13.2 GCU による OpenVMS Galaxy の管理

GCU を使用して Galaxy システムを管理する機能は， 管理操作に関係する各インスタンスの機能に応じて異なります。

GCU は Galaxy のどのインスタンスからも実行できます。 しかし，Galaxy ソフトウェア・アーキテクチャでは， リソースの再割り当てのためにプッシュ・モデルを実装しています。つまり，プロセッサを再割り当てするには，プロセッサを現在所有しているインスタンスで再割り当てコマンド機能を実行しなければなりません。GCU はこの必要条件を認識しており，1 つ以上の通信パスを使用して，再割り当て要求を所有者インスタンスに送信しようとします。DCL はこの必要条件を認識していないため， DCL を使用してリソースを再割り当てする場合は， SYSMAN を使用するか，または個別にログインした端末を使用して， 所有者インスタンスでコマンドを実行しなければなりません。

GCU では，SYSMAN とその基礎になっている SMI_Server プロセスの使用が歓迎され， Galaxy の他のインスタンスにコマンド・パスが提供されます。しかし， SMI_Server では，コマンド環境が共通のセキュリティ・ドメイン内に存在するように，インスタンスはクラスタ内に存在していなければなりません。 しかし，Galaxy インスタンスはクラスタ接続されていない可能性があります。

SMI_Server に適したコマンド・パスをシステムが提供できない場合は， GCU は DECnet タスク間通信を使用しようとします。 このためには，参加するインスタンスが DECnet を実行していなければならず， 参加する各 Galaxy インスタンスは SYSTEM アカウントに対してプロキシを 設定しておく必要があります。

13.2.1 独立インスタンス

1 つ以上のインスタンスが Galaxy 共用コミュニティのメンバにならないように， Galaxy システムを定義することができます。これを **独立インスタンス** と呼びます。独立インスタンスは GCU に認識されます。

これらの独立インスタンスも CPU の再割り当てに参加することができます。 独立インスタンスは共用メモリや関連サービスを利用できません。

13.2.2 分離されたインスタンス

インスタンスをクラスタ接続せず，プロキシ・アカウントを設定せず， DECnet 機能も割り当てないことが可能です。 このようなインスタンスを **分離されたインスタンス** と呼びます。このようなインスタンスは GCU に認識され，CPU を再割り当てすることができます。 分離されたインスタンスからリソースを再割り当てする場合は，分離されたインスタンスのコンソールから行わなければなりません。

13.2.3 必要な PROXY アクセス

GCU が管理アクションを実行しなければならない場合は， 最初に SYSMAN ユーティリティを使用しようとします。 SYSMAN では，関係するインスタンスが同じクラスタ内に存在しなければなりません。 この条件が満たされない場合は，GCU は DECnet タスク間通信を使用しようとします。 この機能を利用するには，関係する各インスタンスにイーサネット装置，DECnet 機能，ターゲット・インスタンスでの適切なプロキシ・アクセスが必要です。

たとえば，クラスタ接続されていない 2 インスタンス構成について考えてみましょう。 インスタンス 0 で GCU が実行されており， ユーザが CPU をインスタンス 1 からインスタンス 0 に再割り当てしようとすると， 実際の再割り当てコマンドはインスタンス 1 で実行しなければなりません。 このために，ファイル SYS$MANAGER:GCU$ACTIONS.COM 内の GCU のアクション・プロシージャは，インスタンス 1 の SYSTEM アカウントに対して DECnet タスク間接続を確立しようとします。 このためには，インスタンス 1 に対して，インスタンス 0 の SYSTEM アカウントへのプロキシ・アクセスが許可されている必要があります。 確立された接続を使用して，インスタンス 0 のアクション・プロシージャは， パラメータをインスタンス 1 の対応するアクション・プロシージャに渡し，そのアクション・プロシージャは操作をローカル操作として取り扱います。

GCU アクション・プロシージャは，それがシステム管理者によって使用されるものと想定します。したがって，アクション・プロシージャ・ファイル SYS$MANAGER:GCU$ACTIONS.COM で，SYSTEM アカウントが使用されます。相手のインスタンスの SYSTEM アカウントへのアクセスを許可するには，インスタンス 1 でプロキシを設定しなければなりません。

プロキシ・アクセスを設定するには

1. DCL プロンプトに対して，次のコマンドを入力します。

```
$ SET DEFAULT SYS$SYSTEM
$ RUN AUTHORIZE
```

2. プロキシ処理がまだ有効に設定されていない場合は，次のコマンドを入力して有効にします。

```
UAF> CREATE/PROXY
UAF> ADD/PROXY instance::SYSTEM SYSTEM
UAF> EXIT
```

instance を，アクセスを許可しているインスタンスの名前に置き換えます。GCU を実行しているインスタンスから管理する各インスタンスに対して，これらの操作を実行します。たとえば，典型的な 2 インスタンス Galaxy で，インスタンス 0 でのみ GCU を実行している場合は，インスタンス 0 に対して，インスタンス 1 でのみプロキシ・アクセスを追加する必要があります。GCU をインスタンス 1 でも実行する予定がある場合は，インスタンス 1 に対してインスタンス 0 でプロキシ・アクセスを追加する必要があります。3 インスタンス Galaxy システムでは，制御する各インスタンスの各組み合わせに対して，プロキシ・アクセスを追加しなければなりません。この理由から，GCU は常にインスタンス 0 から実行するようにしてください。

SYSTEM アカウントを使用する必要はありません。アカウントを変更するには，関係する各インスタンスで SYS$MANAGER:GCU$ACTIONS.COM を変更する必要があります。タスク間接続を設定する行を探し，SYSTEM アカウント名を適切な名前に変更します。

選択したアカウントには OPER，SYSPRV，CMKRNL 特権が必要です。また，このアカウントに対して自分のインスタンスへの必要なプロキシ・アクセスを追加する必要もあります。

13.3 Galaxy 構成モデル

GCU は完全にプログラミング可能な表示エンジンです。GCU は，ルールセットを使用して，システム・コンポーネントの動作や適切な属性に関する知識を取得します。この特別な構成に関する知識を利用して，GCU はシステム・コンポーネント間の関係を表すモデルを組み立てます。GCU は，コンソール・ファームウェアが構築した構成モデルを解析することで，現在のシステム構造を解析します。この構造は Galaxy 構成ファイルと呼ばれ，メモリに格納され，必要に応じてファームウェアおよび OpenVMS エグゼクティブ・ルーチンで更新され，現在のシステム構成および状態が正確に反映されます。

GCU は構成ファイルのバイナリ表現を単純な ASCII 表現に変換し，拡張します。この表現はオフライン・モデルとしてファイルに格納できます。GCU は後でオフライン・モデルを再ロードし，モデルに対応するようにシステム構成を変更できます。アクティブ・モデルを表示している場合も，オフライン・モデルを表示している場合も，現在の構成はいつでもオフライン Galaxy 構成モデル (.GCM) ファイルとして保存できます。

オフライン・モデルを現在のシステム構成として使用するには，モデルをロードし，そのモデルに設定しなければなりません。モデルに設定するには，[Engage] ボタンをクリックします。GCU は現在の構成ファイルをスキャンし，それをモデルと比較し，モデルに設定するのに必要な管理アクションの一覧を作成します。GCU はこの一覧を最終確認のために表示します。ユーザが承認すると，GCU はアクションを実行し，現在のシステム構成および状態を反映するように，モデルは設定された状態になります。

モデルの設定を解除すると，GCU はただちに CPU とインスタンスをオフラインとしてマークします。 その後，モデルを変更することができ， モデルを保存したり，モデルに再び設定することができます。通常，ビジネスにとって役立つことが証明されたモデルだけを若干数保存します。 これらのモデルは，システム管理者または適切な特権が与えられたユーザ， または DCL コマンド・プロシージャを使用して設定することができます。

13.3.1 アクティブ・モデル

GCU は 1 つのアクティブ・モデルを管理します。 このモデルは常にメモリ内の構成ファイルから作成されます。 構成ファイルは Galaxy コンソールまたは Alpha システムのファイル・ベースのシングル・インスタンス Galaxy から取得できます。どこから取得した場合でも， コンソール・コールバックがファイルの保全性を管理します。

GCU は Galaxy イベント・サービスを使用して， 構成がいつ変更されたかを判断します。 構成が変更されると，GCU は構成ファイルを解析し， 現在のシステムを反映するようにアクティブ・モデルを更新します。アクティブ・モデルは，オフライン・モデルとして保存することを選択しない限り， ファイルに保存されません。 通常，アクティブ・モデルは追加モデルを作成するための基礎になります。モデルを作成する場合，必要に応じてオフライン・モデルに設定できるように， オンラインで作成するのが最適です。

13.3.2 オフライン・モデル

GCU はオフライン Galaxy 構成モデルをいくつでもロードすることができ， それが特定のシステム・ハードウェア用に作成されたものである場合， 自由に切り換えることができます。モデルの表現は単純な ASCII データ定義フォーマットです。

モデル・ファイルを ASCII 形式で編集する必要はありません。 GCU モデルとルールセットは Galaxy 構成言語 (GCL) と呼ぶ単純な言語に準拠しています。この言語は新しい Galaxy の発展に伴って，必要に応じて進歩していくでしょう。モデルとルールセット・ファイルを直接作成する場合は，このことに注意する必要があります。モデルを誤って壊した場合は，いつでも別のモデルを作成できます。ルールセットを壊した場合は， OpenVMS Galaxy Web サイトから別のルールセットをダウンロードする必要があります。

13.3.2.1 例: オフライン・モデルの作成

オフライン Galaxy 構成モデルを作成するには

1. Galaxy システムをブートし，システム・アカウントにログインし，GCU を実行します。
2. デフォルト設定で，GCU はアクティブ・モデルを表示します。
3. [Engage] ボタンをクリックして (トグルします)， アクティブ・モデルを設定解除状態にします。
4. システムにいくつかのセカンダリ CPU がある場合は， 一部の CPU を別の Galaxy インスタンスにドラッグ・アンド・ドロップします。
5. [Model] メニューの [Save Model] を選択して，モデルを保存します。 モデルに適切な名前を付け，.GCM 拡張子を付けます。 モデル名は CPU の割り当てがわかるような名前にしておくと便利です。 たとえば，インスタンス 0 に 1 つの CPU が割り当てられ， インスタンス 1 に 7 つの CPU が割り当てられているシステムの場合は， G1x7.GCM という名前を付け， 2 つの各インスタンスに 4 つの CPU が割り当てられているシステムの場合は，G4x4.GCM という名前を付けておくとわかりやすいでしょう。 名前の付け方は任意ですが，必ず .GCM 拡張子を付けてください。

   必要に応じて，モデルはいくつでも作成でき，保存できます。

オフライン・モデルに設定するには

1. GCU を実行します。
2. デフォルト設定で，GCU はアクティブ・モデルを表示します。 アクティブ・モデルは閉じることができ，そのままにしておくこともできます。
3. [Model] メニューから [Open Model] を選択して，適切なモデルをロードします。

4. 適切なモデルを探して選択し，[OK] をクリックします。 モデルがロードされ，オフラインで表示され，設定解除状態になります。
5. [Engage] ボタンをクリックして，再びモデルに設定します。
6. GCU はモデルに設定するのに必要な管理アクションを表示します。 それらのアクションを承認する場合は，[OK] をクリックします。GCU は管理アクションを実行し，モデルはアクティブで， 設定されたモデルとして表示されます。

13.4 GCU チャートの使用

GCU には，かなり大量の構成データが格納されます。 複雑な Galaxy 構成の場合は，ファイルが非常に大きくなる可能性があります。 GCU がシステムに関するすべての情報を表示した後， 表示の内容は非常に複雑になります。この問題を回避するために， GCU では Galaxy チャートを使用します。チャートはさまざまなコンポーネント， 装置，相互接続の表示を制御するためのマスクの集合です。コンポーネント階層構造の中で，各チャートには，そのチャートで指定されたコンポーネントだけが表示されます。 別のチャートを選択すると，別のコンポーネント・サブセットが表示されます。

デフォルト設定では，GCU は 5 つのあらかじめ構成されたチャートを使用します。

- Physical Structure チャート (13.4.2 項「Physical Structure チャート」)
- Logical Structure チャート (13.4.3 項「Logical Structure チャート」)
- Memory Assignment チャート (13.4.4 項「Memory Assignment チャート」)
- CPU Assignment チャート (13.4.5 項「CPU Assignment チャート」)
- IOP Assignment チャート (13.4.6 項「IOP Assignment チャート」)
- Failover Target チャート (13.4.7 項「Failover Target チャート」)

各チャートは，固有のコンポーネント関係を表示するように設計されています。 一部の GCU コマンド操作は，特定のチャートの内部でのみ実行できます。たとえば， Physical Structure チャートの内部から CPU を再割り当てすることはできません。 Physical Structure チャートには， Galaxy インスタンス・コンポーネントは表示されないので， CPU のドラッグ・アンド・ドロップの宛先がありません。 同様に，Logical Structure チャートを表示しているときにはホット・スワップ・インクワイアリ操作を実行することはできません。デバイスのホット・スワップは物理的なタスクなので， 論理チャートに表示することはできません。 チャートはユーザが変更できるので， GCU ではそのメニューおよびコマンド操作を特定のチャートに制限していません。 場合によっては， 適切なチャートを選択するのに役立つように，GCU は情報メッセージを表示します。

13.4.1 コンポーネントの識別と表示プロパティ

各コンポーネントには固有の識別子があります。 この識別子は，CPU ID などの場合は単純な連続番号であり， I/O アダプタの場合は物理バックプレーン・スロット番号であり， メモリ装置の場合は物理アドレスです。GCU は各種類のコンポーネントに形状と色も割り当てます。可能な場合は，GCU はさらに動作中のシステムから収集した補足情報を利用して，各コンポーネントを区別します。

各コンポーネントの表示プロパティは， Galaxy 構成ルールセット (SYS$MANAGER:GALAXY.GCR) の内部で割り当てられます。 ウィンドウの色や表示テキストのスタイルなどの特定の表示プロパティをカスタマイズする場合を除き，このファイルは編集すべきではありません。

各コンポーネントに関して表示されるテキストもカスタマイズできます。各種類のコンポーネントには，その外観， 内容，相互関係を示す文がルールセットで割り当てられます。

1 つの役立つ機能として，どのテキストを画面上で各コンポーネント・タイプに表示するかを選択する機能があります。ルールセット内で装置宣言を使用すると， 表示テキスト文を構成するテキストとパラメータを指定できます。 現在のズーム係数で完全なテキストを表示できない場合は，この表示テキストのサブセットが表示されます。このサブセットをニーモニックと呼びます。 ニーモニックは任意のテキストやパラメータを含むように変更できます。

13.4.2 Physical Structure チャート

Physical Structure チャートには，システム内の物理ハードウェアが表示されます。 チャートのルート (一番上) に表示される大きな長方形のコンポーネントは， 物理システム・キャビネット自体を表します。通常，ルートの下にモジュール，スロット，アレイ，アダプタなどの物理コンポーネントがあります。表示されるコンポーネントの種類とコンポーネント階層構造の深さは，各ハードウェア・プラットフォームのコンソール・ファームウェアで提供される サポート・レベルに応じて異なります。 Alpha システムのシングル・インスタンス Galaxy を表示している場合は， コンポーネントの小さなサブセットだけが表示されます。

一般に，コンソール・ファームウェアは構成可能な装置レベル， 通常は第 1 レベルの I/O アダプタまたはその少し下までのコンポーネントだけを表示します。 GCU や Galaxy コンソール・ファームウェアにとってすべての装置をマッピングすることが目標なのではなく，Galaxy 構成管理にとって関心のある装置だけをマッピングすることが必要なのです。

Physical Structure チャートは， システム内のコンポーネント全体を表示するのに役立ちます。しかし，コンポーネントの論理パーティションは表示されません。

Physical Structure チャートでは次の操作が可能です。

- システム・コンポーネントのパラメータを確認できます。
- 修理するコンポーネントを突き止める方法を判断するために，ホット・スワップ・インクワイアリを実行できます。
- 最適な性能を実現できるように，ハードウェア・プラットフォームで特定の最適化が行われているかどうか判断するために， 最適化オーバーレイを適用できます。たとえば，マルチ CPU モジュールでは， 共通のモジュールにあるすべての CPU が同じ Galaxy インスタンスに割り当てられた場合，最適な性能を実現できます。
- Galaxy または特定の Galaxy インスタンスをシャットダウンまたはリブートできます。

13.4.2.1 ハードウェア・ルート

Physical Structure チャートの一番上のコンポーネントをハードウェア・ルート (HW_Root) と呼びます。 すべての Galaxy システムにハードウェア・ルートが 1 つずつあります。 これは，物理的に設置されているマシンであると考えると便利です。物理装置の場所がコンポーネント階層構造にない場合は， ハードウェア・ルートの子として表示されます。 子コンポーネントは，マシンがパーティション分割または論理的に定義されるときに， 階層構造内の他の装置に割り当てることができます。

注意:

自動レイアウト・モードに設定されている場合，どのチャートでもルート・インスタンスをクリックすると， 自動レイアウト操作が実行されます。

13.4.2.2 所有権オーバーレイ

[Windows] メニューから [Ownership Overlay] を選択すると， さまざまなコンポーネントの所有者の初期関係が表示されます。これらの関係は，パワー・サイクルの後，コンポーネントを所有するインスタンスを示します。システムがブートされた後，マイグレート可能なコンポーネントは， 所有者を動的に変更することができます。 初期所有権を変更するには，コンソール環境変数を変更しなければなりません。

所有権オーバーレイは Physical Structure チャートと Failover Target チャートでは無効です。

13.4.3 Logical Structure チャート

Logical Structure チャートには，Galaxy コミュニティとインスタンスが表示され， Galaxy を形成するすべての関係が最もわかりやすく示されます。これらのコンポーネントの下に，現在所有しているさまざまな装置が表示されます。 Logical Structure チャートと Physical Structure チャートでは， 所有権に重要な違いがあります。Galaxy では，パーティション分割， または動的に再構成できるリソースには 2 つの異なる所有者があります。

所有者とは，システムの電源がオンになった後，装置の電源がどこでオンになるかを記述するものです。この値はバス・プローブ・プロシージャと Galaxy 環境変数の解釈時に，コンソール・ファームウェアによって判断されます。所有者の値はコンソールの不揮発性メモリに格納されるので，パワー・サイクルの後，復元することができます。

CURRENT_OWNER は，特定の時点における装置の所有者を示します。 たとえば，CPU はインスタンス間で自由に再割り当てすることができます。その場合，CURRENT_OWNER の値は変化しますが， owner の値は LP_CPU_MASK# 環境変数によって設定された値のまま変化しません。

Logical Structure チャートには，CURRENT_OWNER の関係が表示されます。 変化しない所有者の関係を表示するには， [Window] メニューから [Ownership Overlay] を選択します。

この後の項では，Logical Structure チャートのコンポーネントを説明します。

13.4.3.1 ソフトウェア・ルート

Logical Structure チャートの一番上のコンポーネントはソフトウェア・ルート (SW_Root) です。すべての Galaxy システムにソフトウェア・ルートが 1 つずつあります。 物理装置に特定の所有者がない場合は，ソフトウェア・ルートの子として表示されます。子コンポーネントは，マシンが論理的に定義されるときに， 階層構造内の他の装置に割り当てることができます。

注意:

自動レイアウト・モードが設定されている場合，どのチャートでもルート・インスタンスをクリックすると， 自動レイアウト操作が実行されます。

13.4.3.2 未割り当てリソース

すべての装置をパーティションに割り当てずに，Galaxy パーティションを構成することができ，1 つ以上のパーティションを定義するだけで， 初期化しないことも可能です。 どちらの場合も， システムのブート時に一部のハードウェアは未割り当ての状態になります。

コンソール・ファームウェアは，未割り当てリソースを次の方法で取り扱います。

- 未割り当て CPU はパーティション 0 に割り当てられます。
- 未割り当てメモリは無視されます。

システム・ブートの後，割り当てられていない装置はソフトウェア・ルート・コンポーネントに割り当てられているものとして表示され， アクセスできなくなります。

13.4.3.3 コミュニティ・リソース

共用メモリなどのリソースは，共用コミュニティ内のすべてのインスタンスからアクセスできます。 したがって，共用メモリの場合，コミュニティ自体が所有者であると考えられます。

13.4.3.4 インスタンス・リソース

特定のインスタンスによって現在所有されているリソースや永久的に所有されているリソースは，インスタンス・コンポーネントの子として表示されます。

13.4.4 Memory Assignment チャート

Memory Assignment チャートは，Galaxy インスタンス間のメモリの割り当てとパーティションを示します。このチャートには， ハードウェア・コンポーネント (アレイ，コントローラなど) とソフトウェア・コンポーネント (メモリ・フラグメント) の両方が表示されます。

現在の Galaxy ファームウェアおよびオペレーティング・システム・ソフトウェアでは，メモリの動的な再構成はサポートされません。 したがって，Memory Assignment チャートには，Galaxy インスタンス間でコンソールによってメモリ・アドレス空間がどのようにパーティション分割されているかが表示されます。 この情報はシステム・アプリケーションをデバッグする場合や， 可能な構成の変更方法を確認するときに役立ちます。

この後の項では，メモリ・フラグメントの説明をします。

13.4.4.1 コンソール・フラグメント

コンソールには 1 つ以上の小さなメモリ・フラグメントが必要です。 通常，コンソールは各パーティションの下位アドレスに約 2 MB のメモリを割り当てます。これはハードウェア・プラットフォームやファームウェアのリビジョンに応じて異なります。さらに，一部のコンソールでは，メモリ・ビットマップを格納するために，各パーティションの上位アドレス空間に小さなフラグメントを割り当てます。コンソール・ファームウェアは適切なメモリ・アライメントが行われるように， 追加フラグメントを作成しなければならないことがあります。

13.4.4.2 プライベート・フラグメント

各 Galaxy インスタンスは，OpenVMS をブートするために少なくとも 64 MB の プライベート・メモリ (コンソール・フラグメントを含む) を必要とします。 このメモリはシングル・フラグメントで構成することができ，コンソール・ファームウェアは適切なメモリ・アライメントが行われるように，追加プライベート・フラグメントを作成しなければならないこともあります。

13.4.4.3 共用メモリ・フラグメント

OpenVMS Galaxy を作成するには， 少なくとも 8 MB の共用メモリを割り当てなければなりません。 つまり，OpenVMS Galaxy にとって必要最低限のメモリは， 実際には 72 MB (シングル・インスタンスのための 64 MB と，共用メモリのための 8 MB) です。

13.4.5 CPU Assignment チャート

CPU Assignment チャートには，Galaxy インスタンス間で CPU を再割り当てするのに必要な最小のコンポーネントが表示されます。 このチャートは非常に大きな Galaxy 構成を取り扱うときに便利です。

13.4.5.1 プライマリ CPU

各プライマリ CPU は六角形ではなく，楕円で表示されます。これは，プライマリ CPU を再割り当てしたり，停止することができないことを示すためです。 プライマリ CPU をドラッグ・アンド・ドロップしようとすると，GCU はステータス・バーにエラー・メッセージを表示し，操作を実行しません。

13.4.5.2 セカンダリ CPU

セカンダリ CPU は六角形で表示されます。セカンダリ CPU は， Logical Structure チャートまたは CPU Assignment チャートでインスタンス間で再割り当てすることができます。 その場合，CPU を適切なインスタンスにドラッグ・アンド・ドロップします。 現在所有しているインスタンスと同じインスタンスに CPU をドロップすると， CPU は停止され，再起動されます。

13.4.5.3 Fast Path およびアフィニティされた CPU

Fast Path を使用する装置に対応づけられた CPU を再割り当てすると，装置に対する対応づけは他の CPU に移動し，CPU の再割り当ては正常に完了します。 CPU に現在，プロセス・アフィニティ割り当てがある場合，CPU を再割り当てすることはできません。

OpenVMS Fast Path 機能の詳細については，『OpenVMS I/O User's Reference Manual』を参照してください。

13.4.5.4 失われた CPU

セカンダリ CPU は，まだブートされていないインスタンス (パーティション) に再割り当てすることができます。

同様に，Galaxy 共用コミュニティのメンバとして構成されていないインスタンスに CPU を再割り当てすることができます。 この場合，現在の所有者インスタンスから CPU をプッシュす

ることができますが，独立インスタンス (個別のセキュリティ・ドメイン) にログインし，CPU を現在の所有者に再割り当てしない限り，元に戻すことはできません。

インスタンスが Galaxy 共用コミュニティの一部であるのか，独立インスタンスであるのかとは無関係に，そのインスタンスは Galaxy 構成ファイルに格納されます。したがって，GCU はそのインスタンスを表示することができます。

13.4.6 IOP Assignment チャート

IOP Assignment チャートには，I/O モジュールと Galaxy インスタンスの現在の関係が表示されます。使用しているハードウェア・プラットフォームに応じて，Alpha システム上のシングル・インスタンス Galaxy は，このチャートに I/O モジュールを表示しないことがあります。

13.4.7 Failover Target チャート

Failover Target チャートには，シャットダウンや障害が発生したときに，各プロセッサが他のインスタンスに自動的にフェールオーバする方法が表示されます。さらに，このチャートには，各 CPU のオートスタート・フラグの状態も表示されます。

各インスタンスに対してフェールオーバ・オブジェクトの集合が表示され，フェールオーバ可能な CPU が示されます。デフォルト設定では，フェールオーバ関係は設定されず，すべてのオートスタート・フラグが設定されます。

特定の CPU の自動フェールオーバを設定するには，CPU のフェールオーバ先のインスタンスに適切なフェールオーバ・オブジェクトをドラッグ・アンド・ドロップします。インスタンスが所有するすべての CPU のフェールオーバ関係を設定するには，CPU のフェールオーバ・ターゲットとなるインスタンスの上にインスタンス・オブジェクトをドラッグ・アンド・ドロップします。

個々のフェールオーバ・ターゲットを消去するには，フェールオーバ・オブジェクトを所有者インスタンスにドラッグ・アンド・ドロップします。すべてのフェールオーバ関係を消去するには，インスタンス・オブジェクトを右クリックして，[Parameters & Commands] ダイアログ・ボックスを表示し，[Commands] ボタンをクリックし，[Clear ALL failover targets?] をクリックし，[OK] をクリックします。

デフォルト設定では，フェールオーバ操作が発生すると，CPU がターゲット・インスタンスに到着した後，自動的に起動されます。このオートスタート機能は，各フェールオーバ・オブジェクトまたは各インスタンス・オブジェクトに対して，[Parameters & Commands] ダイアログ・ボックスのオートスタート・コマンドを使用して制御できます。Failover Target チャートには，オートスタート・フラグの状態が表示されます。オートスタートがセットされている場合は，フェールオーバ・オブジェクトが緑で表示され，オートスタートがクリアされている場合は赤で表示されます。

フェールオーバおよびオートスタート管理の現在の実装には，次の制限事項がありますので注意してください。

- フェールオーバおよびオートスタートの設定は，システム・ブートのたびに失われます。したがって，システムをリブートするときに，必ずモデルに再設定する必要があります。このためには，前に保存しておいた構成モデルを呼び出します。手動で適切なモデルを復元するか，システム起動時にコマンド・プロシージャを使用してください。
- GCU は現在，インスタンスがクラスタ接続されていない限り，GCU が実行されているインスタンス以外のインスタンスのオートスタートおよびフェールオーバ関係を判断することができません。
- GCU は現在，GCU の別の実行可能コピーや DCL コマンドから行われた フェールオーバまたはオートスタート状態の変更に対応することができません。この状態が変更された場合，アクティブ・モデルをいったん終了して開いた場合にだけ，GCU は表示を更新します。

13.5 コンポーネント・パラメータの表示

各コンポーネントには表示可能なパラメータがあり，場合によっては変更することも可能です。 コンポーネントのパラメータを表示するには， カーソルを適切なコンポーネントの上に置き，マウスの右ボタンをクリックして，ポップアップ・メニューから [Parameters] を選択します。 また，コンポーネントを選択し，[Components] メニューから [Parameters] を選択することもできます。

パラメータの単位が変換される場合，表示単位を変更すると，表示と，現在表示されているパラメータ・ダイアログ・ボックスが更新されます。他のパラメータはシステム・コンポーネントのスナップショットであり， 動的に更新されません。これらのパラメータを変更した場合は， [Parameters] ダイアログ・ボックスをいったん閉じて再び開くことで， 値を更新しなければなりません。

13.6 コンポーネント・コマンドの実行

コンポーネントの [Parameters] ダイアログ・ボックスには， コマンド・ページも表示されることがあります。その場合は， ダイアログ・ボックスの上部に表示される [Commands] ボタンをクリックすることで，コマンドにアクセスできます。大部分のコマンドは，トグル・ボタンをクリックして， [OK] または [Apply] ボタンをクリックすることで実行されます。 また，情報を入力しなければならないコマンドや，リストやオプション・メニューから値を選択しなければならないコマンドもあります。複数のコマンドを選択すると，コマンドは上から下に順に実行されます。 論理的に正しいコマンド・シーケンスを選択してください。

13.7 GCU メニューのカスタマイズ

システム管理者は SYS$MANAGER:GCU$CUSTOM.GCR というファイルを作成することで，GCU メニューとメニュー・エントリを拡張し，カスタマイズできます。 このファイルには，次の例に示す形式のメニュー文だけを格納しなければなりません。 GCU$CUSTOM.GCR ファイルはオプションです。 このファイルはオペレーティング・システムのアップグレード時に変更されません。

```
FORMAT EXAMPLE:

  MENU "Menu-Name" "Entry-Name" Procedure-type "DCL-command"

      * Menu-Name - A quoted string representing the name of the
                    pulldown menu to add or extend.

      * Entry-Name - A quoted string representing the name of the
                     menu entry to add.

      * Procedure-type - A keyword describing the type of procedure
                     to invoke when the menu entry is selected.

        Valid Procedure-type keywords include:

        COMMAND_PROCEDURE - Executes a DCL command or command file.
        SUBPROC_PROCEDURE - Executes a DCL command in subprocess context.

      * DCL-command - A quoted string containing a DCL command statement
                      consisting of an individual command or invokation
                      of a command procedure.
```

他のインスタンスで実行するプロシージャを作成するには， GCU が SYS$MANAGER:GCU$ACTIONS.COMT で使用するものと同様の SYSMAN またはタスク間方式を利用するコマンド・プロシージャを書きます。 GCU$ACTIONS.COM は拡張できますが，このファイルはオペレーティング・システムのアップグレード時に変更される可能性があります。

```
EXAMPLE MENU STATEMENTS (place in SYS$MANAGER:GCU$CUSTOM.GCR):

// GCU$CUSTOM.GCR - GCU menu customizations
// Note that the file must end with the END-OF-FILE statement.
//
MENU "Tools" "Availability Manager" SUBPROC_PROCEDURE "AVAIL/GROUP=DECamds"
MENU "Tools" "Create DECterm" COMMAND_PROCEDURE  "CREATE/TERM/DETACH"
MENU "DCL"   "Show CPU"       COMMAND_PROCEDURE  "SHOW CPU"
MENU "DCL"   "Show Memory"    COMMAND_PROCEDURE  "SHOW MEMORY"
MENU "DCL"   "Show System"    COMMAND_PROCEDURE  "SHOW SYSTEM"
MENU "DCL"   "Show Cluster"   COMMAND_PROCEDURE  "SHOW CLUSTER"
END-OF-FILE
```

13.8 HP Availability Manager による OpenVMS Galaxy の監視

Availability Manager ソフトウェアは，Galaxy システムをリアルタイムで表示できる便利なツールです。 Availability Manager はローカル・エリア・ネットワークにある 1 台のワークステーションまたは PC からすべての Galaxy インスタンスを監視できます。 Availability Manager は，システムから可用性に関するデータを定期的に収集するカスタム OpenVMS ドライバ (RMDRIVER) を利用します。 この情報は，下位レベルのイーサネット・プロトコルを使用して，Availability Manager Data Analyzer に戻されます。

Data Analyzer は，システムの可用性属性をさまざまな表示やグラフで示します。 さらに，Availability Manager は，認識する多くの条件のいずれかを検出すると，そのことをユーザに通知し，その問題を解決するために一連のソリューション (フィックスと呼びます) を提供します。

各 OpenVMS システムには Availability Manager Data Collector (RMDRIVER) が インストールされています。 コネクタを有効にするには，SYSTARTUP_VMS.COM の内部でスタートアップ・プロシージャを実行するか，監視する各 Galaxy インスタンスで手動でスタートアップ・プロシージャを実行しなければなりません。 データ・コレクタを起動または停止するには，次のコマンドを入力します。

```
$ @SYS$STARTUP:AMDS$STARTUP START
```

または

```
$ @SYS$STARTUP:AMDS$STARTUP STOP
```

コレクタを起動する前に，Galaxy のグループ名を指定する必要があります。 その場合は，SYS$MANAGER:AMDS$LOGICALS.COM ファイルを編集してください。 このファイルには，グループ名を宣言するための文が格納されています。 固有の名前を選択し，各 Galaxy インスタンス上でこのファイルに同じグループ名が格納されるようにしてください。

OpenVMS エンジニアリング・グループは，Availability Manager を使用するときに，各 Galaxy インスタンスに対して，[System Overview] ウィンドウと，[Node Summary Page] の [CPU Modes Summary Page] を表示すると便利なことに気付きました。 それぞれの要件に応じて，多くの追加ビューを監視できます。 Availability Manager の詳細については，『Availibility Manager User's Manual』を参照してください。

13.9 CPU Load Balancer プログラムの実行

OpenVMS Galaxy CPU Load Balancer プログラムは OpenVMS Galaxy にあるインスタンス内の CPU リソースを直接再割り当てする特権アプリケーションです。

GCU からこのプログラムを実行する方法については， 付録 A「OpenVMS Galaxy CPU Load Balancer プログラム」 を参照してください。

13.10 インスタンスの作成

OpenVMS Galaxy ソフトウェア・アーキテクチャの現在の実装では，使用する予定の Galaxy インスタンスをあらかじめ定義しておく必要があります。その場合はコンソール環境変数を使用します。 Galaxy 環境変数の詳細については，本書の該当する節を参照してください。

13.11 インスタンスの破棄

Galaxy インスタンスを破棄するには，インスタンスをシャットダウンし，コンソール環境変数またはプロシージャを使用してリソースを再割り当てし，必要に応じて新しいリソースを取得するインスタンスをリブートします。

13.12 シャットダウンおよびリブート・サイクル

CPU などのリソースは，関係するインスタンスがブートされた後， 動的に再割り当てすることができます。I/O モジュールなどの静的に割り当てられたリソースを再割り当てするには，適切なコンソール・コマンドを実行した後， 関係するインスタンスをシャットダウンし，リブートしなければなりません。

13.13 オンライン・モデルとオフライン・モデル

GCU では，アクティブ (オンライン) Galaxy 構成モデルまたは非アクティブ (オフライン) Galaxy 構成モデルを表示し，会話することができます。 構成表示がアクティブ・システムのモデルを表す場合は， GCU は色とテキストを使用して，CPU とインスタンスの状態を表示します。 構成モデルがこの方法で設定された場合は， ドラッグ・アンド・ドロップを使用してアクティブ・システムと会話することができます。この操作モードを正しく表現すると，「設定されているオンライン・モデルとの会話」になります。

GCU ユーザは，設定が解除されているモデル， つまり複数のオフライン・モデルと会話することもできます。 オフライン・モデルはファイルに保存したり， ファイルからロードすることができます。また，オフライン・モデルはアクティブ・オンライン・モデルから作成することもできます。 その場合は，アクティブ・オンライン・モデルが表示されているときに，[Engage] ボタンをクリックして， モデルの設定を解除します。[Engage] ボタンの表示状態の他に， GCU は CPU とインスタンスのオンライン属性とオフライン属性を， 色やテキストで示します。オフライン・モデルで指示されたドラッグ・アンド・ドロップ操作は，単純な編集機能として解釈されます。 これらはモデルの内部構造を変更しますが，アクティブ・システムには影響しません。

オフライン・モデルが設定されると，GCU はモデルの構造をアクティブ・システムの構造と比較します。一致する場合は，オフライン・モデルに設定され， 新しいオンライン状態が色とテキストによって示されます。 一致しない場合は，GCU は提案されたモデルと一致するようにアクティブ・システムを変更するために，どのような管理アクションが必要かを判断します。 判断した管理アクションの一覧をユーザに示し， そのアクションを実行してもよいかどうか質問します。ユーザがアクションの実行を認めないと，モデルはオフラインおよび設定解除状態のままになります。 ユーザがアクションを認めると， GCU は管理アクションを実行し，作成されたモデルはオンラインおよび設定状態のモデルとして表示されます。

13.14 GCU システム・メッセージ

次のシステム・メッセージは，エラーが発生した際に GCU によって表示されます。

- **%GCU-E-SUBPROCHALT, Subprocess halted; See GCU.LOG.**

  GCU が起動したユーザ定義サブプロセスはエラー状態で終了しました。詳細は GCU.LOG ファイルを参照してください。

- **%GCU-S-SUBPROCTERM, Subprocess terminated**

  GCU が起動したユーザ定義サブプロセスは終了しました。

- **%GCU-I-SYNCMODE, XSynchronize activated**

  GCU は X-windows 同期モードが有効な状態で起動されました。 これは開発モードであり，通常は使用されません。

- **%GCU-W-NOCPU, Unable to locate CPU**

  不明の CPU に関係するマイグレーション・アクションが開始されました。 これは，現在のシステムにとって不正な CPU 識別子を含むモデルに設定したために発生している可能性があります。

- **%GCU-E-NORULESET, Ruleset not found:**

  GCU は SYS$MANAGER:GALAXY.GCR から Galaxy 構成ルールセットを見つけることができませんでした。 このファイルの新しいバージョンは OpenVMS Galaxy Web ページからダウンロードできます。

- **%GCU-E-NOMODEL, Galaxy configuration model not found:**

  指定された Galaxy 構成モデルが見つかりません。 コマンド行に指定したモデル・ファイルが正しいかどうか確認してください。

- **%GCU-W-XTOOLKIT, X-Toolkit Warning:**

  GCU は X-Toolkit 警告を受信しました。警告の種類に応じて，操作を続行できる場合と，できない場合があります。

- **%GCU-S-ENGAGED, New Galaxy configuration model engaged**

  GCU は新しい Galaxy 構成モデルに正しく設定することができました。

- **%GCU-E-DISENGAGED, Unable to engage Galaxy configuration model**

  GCU は新しい Galaxy 構成モデルに設定することができませんでした。 この問題が発生するのは，指定したモデルが現在のシステムにとって不正であるか，他のシステム動作によって，要求されたリソースの割り当てが不可能な場合です。

- **%GCU-E-NODECW, DECwindows is not installed.**

  現在のシステムでは，要求された DECwindows サポート機能が提供されません。

- **%GCU-E-HELPERROR Help subsystem error.**

  DECwindows ヘルプ・システム (Bookreader) でエラーが検出されました。

- **%GCU-E-TOPICERROR Help topic not found.**

  DECwindows ヘルプ・システムで指定されたトピックを見つけることができませんでした。

- **%GCU-E-INDEXERROR Help index not found.**

  DECwindows ヘルプ・システムで指定されたインデックスを見つけることができませんでした。

- **%GCU-E-UNKNOWN_COMPONENT: {name}**

  現在のモデルに不明なコンポーネントの参照が含まれています。これは， モデルまたはルールセットが壊れているために発生している可能性があります。指定されたコンポーネントをルールセット SYS$MANAGER:GALAXY.GCR から検索してください。 見つからない場合は，OpenVMS Galaxy Web サイトから新しいコンポーネントをダウンロードしてください。それでも問題を解決できないときは，問題のあるモデルを削除し，再作成してください。

- **%GCU-I-UNASSIGNED_HW: Found unassigned {component}**

  GCU が現在，どの Galaxy インスタンスにも割り当てられていないハードウェア・コンポーネントを検出しました。このコンポーネントは意図的に未割り当ての状態になっているリソースである可能性があります。 メッセージを確認し，操作を続行するか， または

プライマリ Galaxy コンソールからハードウェア・コンポーネントを割り当て，リブートしてください。

- **%GCU-E-UNKNOWN_KEYWORD: {word}**

  GCU は現在のモデル・ファイルで不明なキーワードを解析しました。 この問題が発生するのは，モデル・ファイルの形式が壊れている場合だけです。 問題のあるモデルを削除し，再作成してください。

- **%GCU-E-NOPARAM: Display field {field name}**

  GCU は現在のモデルで不完全なコンポーネント文を解析しました。 この問題が発生するのは，モデル・ファイルの形式が壊れている場合だけです。問題のあるモデルを削除し，再作成してください。

- **%GCU-E-NOEDITFIELD: No editable field in display.**

  GCU は未定義のコンポーネント・パラメータを編集しようとしました。 この問題が発生するのは，モデル・ファイルの形式が壊れている場合だけです。問題のあるモデルを削除し，再作成してください。

- **%GCU-E-UNDEFTYPE, Undefined Parameter Data Type: {type}**

  GCU はモデル・コンポーネント・パラメータで不明なデータ・タイプを解析しました。この問題が発生するのは，モデル・ファイルの形式が壊れているか，現在のモデルと互換性のないルールセットが使用されているためです。 ルールセット SYS$MANAGER:GALAXY.GCR から問題のあるデータ・タイプを検索してください。 見つからない場合は，OpenVMS Galaxy Web サイトから最新のルールセットをダウンロードしてください。 データ・タイプが見つかった場合は， 問題のあるモデルを削除し，再作成してください。

- **%GCU-E-INVALIDMODEL, Invalid model structure in: {model file}**

  GCUが不正なモデル・ファイルをロードしようとしました。問題のあるモデルを削除し，再作成してください。

- **%GCU-F-TERMINATE Unexpected termination.**

  GCU が致命的な DECwindows イベントを検出しました。

- **%GCU-E-GCTLOOP: Configuration Tree Parser Loop**

  GCUが壊れている構成ツリーを解析しようとしました。この問題は，コンソール・ファームウェアまたはオペレーティング・システム・フォルトの結果発生することがあります。

- **%GCU-E-INVALIDNODE: Invalid node in Configuration Tree**

  GCU が構成ツリーの内部で不正な構造を解析しました。この問題が発生するのは，構成ツリーが壊れているか，ルールセットとコンソール・ファームウェアのリビジョンが一致しないためです。

- **%GCU-W-UNKNOWNBUS: Unknown BUS subtype: {type}**

  GCU が現在の構成ツリーで不明なバス・タイプを解析しました。 この問題が発生するのは，ルールセットとコンソール・ファームウェアのリビジョンが一致しないためです。

- **%GCU-W-UNKNOWNCTRL, Unknown Controller type: {type}**

  GCU が現在の構成ツリーで不明なコントローラ・タイプを解析しました。 この問題が発生するのは，ルールセットとコンソール・ファームウェアのリビジョンが一致しないためです。

- **%GCU-W-UNKNOWNCOMP, Unknown component type: {type}**

  GCU が現在の構成ツリーで不明なコンポーネント・タイプを解析しました。 この問題が発生するのは，ルールセットとコンソール・ファームウェアのリビジョンが一致しないためです。

- **%GCU-E-NOIFUNCTION, Unknown internal function**

  ユーザがルールセット・ファイルを変更し，不明な内部 GCU 機能を指定しました。ルールセットを修正するか，OpenVMS Galaxy Web ページから新しいルールセットをダウンロードしてください。

- **%GCU-E-NOEFUNCTION, Missing external function**

  ユーザがルールセット・ファイルを変更し，不明な外部機能を指定しました。ルールセットを修正するか，OpenVMS Galaxy Web ページから新しいルールセットをダウンロードしてください。

- **%GCU-E-NOCFUNCTION, Missing command function**

  ユーザがルールセット・ファイルを変更し，不明なコマンド・プロシージャを指定しました。ルールセットを修正するか，OpenVMS Galaxy Web ページから新しいルールセットをダウンロードしてください。

- **%GCU-E-UNKNOWN_COMPONENT: {component}**

  GCU が不明なコンポーネントを解析しました。この問題が発生するのは，ルールセットが壊れているか，ルールセットとコンソール・ファームウェアのリビジョンが一致しないためです。

- **%GCU-E-BADPROP, Invalid ruleset DEVICE property**

  GCU が不明なルールセット・コンポーネント文を解析しました。この問題が発生するのは，ルールセットが壊れているためです。OpenVMS Galaxy Web ページから新しいルールセットをダウンロードしてください。

- **%GCU-E-BADPROP, Invalid ruleset CHART property**

  GCU が不正なチャート文を解析しました。この問題が発生するのは，ルールセットが壊れているためです。OpenVMS Galaxy Web ページから新しいルールセットをダウンロードしてください。

- **%GCU-E-BADPROP, Invalid ruleset INTERCONNECT property**

  GCU が不正なルールセット・インターコネクト文を解析しました。この問題が発生するのは，ルールセットが壊れているためです。OpenVMS Galaxy Web ページから新しいルールセットをダウンロードしてください。

- **%GCU-E-INTERNAL Slot {slot detail}**

  GCU がコンポーネント・パラメータから不正なデータ・タイプを検出しました。この問題が発生するのは，ルールセットまたはモデルが壊れているためです。OpenVMS Galaxy Web ページから新しいルールセットをダウンロードしてください。それでも問題を解決できない場合は，問題のあるモデルを削除し，再作成してください。

- **%GCU-F-PARSERR, {detail}**

  GCU がルールセットの解析時に致命的なエラーを検出しました。OpenVMS Galaxy Web ページから新しいルールセットをダウンロードしてください。

- **%GCU-W-NOLOADFONT: Unable to load font: {font}**

  GCU が現在のシステムで指定されたフォントを見つけることができません。その代わりにデフォルト・フォントが使用されます。

- **%GCU-W-NOCOLORCELL: Unable to allocate color**

  GCU がカラーマップ・エントリにアクセスできません。この問題が発生するのは，システムで制限された色だけしかサポートされないためか，非常に多くのグラフィカル・アプリケーションが同時に開かれているためです。

- **%GCU-E-NOGALAXY, This system is not configured as a Galaxy.**

  説明:

ユーザが Galaxy 操作用に構成されていないシステムで CONFIGURE GALAXY/ENGAGE コマンドを実行しました。

対処法:

本書で説明した手順に従って，システムを Galaxy 操作用に構成してください。 シングル・インスタンス Galaxy を実行する場合は，/ENGAGE 修飾子を指定せずに，CONFIGURE GALAXY コマンドを入力し， Galaxy Configuration ユーティリティから提供される指示に従ってください。

- **%GCU-E-ACTIONNOTALPHA GCU actions require OpenVMS Alpha**

  GCU ユーザが OpenVMS VAX システムで Galaxy 構成アクションを起動しようとしました。

- **%GCU-I-ACTIONBEGIN at {time}, on {instance} {mode}**

  この情報メッセージは，指定された Galaxy インスタンスで構成アクションが開始されることを示します。 多くのアクションには，2 つの独立した Galaxy インスタンスのコマンド環境間の協調動作が必要であるため， アクションに関係する各インスタンスに対して 1 つずつ， 合計 2 つのメッセージが表示されることがあります。 mode 引数は，どのインスタンスがローカルであり， どのインスタンスがリモートであるかを示します。

- **%GCU-S-ACTIONEND at {time}, on {nodename}**

  これは Galaxy 構成アクションの後に表示される正常終了メッセージです。 多くのアクションでは，2 つの独立した Galaxy インスタンスのコマンド環境間で協調動作が必要であるため，アクションに関係する各インスタンスに対して 1 つずつ，合計 2 つのメッセージが表示されることがあります。

- **%GCU-S-ACTIONEND, Exiting GCU$ACTIONS on ^Y**

  ユーザが Control-Y を使用して Galaxy 構成アクションを打ち切ったことを示します。

- **%GCU-S-ACTIONEND, Exiting GCU$ACTIONS on error {message}**

  メッセージ引数で示されるエラー状態で Galaxy 構成アクションが終了したことを示します。

- **%GCU-E-ACTIONUNKNOWN no action specified**

  GCU$ACTIONS.COM プロシージャが不正に呼び出されたことを示します。 コマンド・プロシージャが壊れているか，現在のシステムのリビジョンと一致していない可能性があります。

- **%GCU-E-ACTIONNOSIN no source instance name specified**

  GCU$ACTIONS.COM が不正に呼び出されたことを示します。 コマンド・プロシージャが壊れているか， 現在のシステムのリビジョンと一致していない可能性があります。

- **%GCU-E-ACTIONBAD failed to execute the specfied action**

  不明なエラーによって Galaxy 構成アクションが異常終了したことを示します。 表示される関連メッセージを確認し，必要なプロキシ・アカウントが設定されているかどうか調べてください。

- **%GCU-E-INSFPRIVS, Insufficient privileges for attempted operation**

  必要な特権が与えられていないユーザが Galaxy 構成アクションを実行しようとしました。通常，これらのアクションはシステム管理者のアカウントの内部から実行されます。OPER 特権と SYSPRV 特権が必要です。

- **%GCU-E-NCF network connect to {instance} failed**

  現在のインスタンスと指定されたインスタンスの間で DECnet タスク間接続を開こうとしたときに，エラーが発生しました。画面に表示される関連メッセージを確認し，必要なプロキシ・アカウントが設定されているかどうか調べてください。

第14章　Graphical Configuration Manager

この章では，ネットワーク・キットに含まれている HP Graphical Configuration Manager (GCM) for OpenVMS について説明します。 キットから GCM サーバと GCM クライアントをインストールする方法を説明し， GCM クライアント/サーバ・セッションを確立するための基本的な説明を述べます。基本的な GCM セットアップを実行すると，GCM クライアントの「Help」メニューから GCM の詳細情報を得ることができます。

14.1 概要

HP Graphical Configuration Manager (GCM) for OpenVMS は， 移植性のあるクライアント/サーバ・アプリケーションであり， OpenVMS を実行しているパーティション分割された AlphaServer システムの構成を視覚的に表示および制御するための手段となります。 GCM クライアントは Java ベースのアプリケーションであり， Java™ ランタイム環境 (JDK V1.2.2 以上) と TCP/IP ネットワークをサポートする任意のオペレーティング・システムで動作します。 GCM サーバは，1 つ以上の AlphaServer システム上で， パーティション分割されたそれぞれの OpenVMS インスタンスの独立プロセスとして実行されます。GCM を使用すると， GCU とほとんど同じ方法で Galaxy システムの構成と管理を行うことができます。 違いは，GCU は DECwindows Motif アプリケーションであり， GCM は Java ベースのアプリケーションであることです。

注意:
GCM クライアントは， 現時点では Java JDK バージョン 1.3 以降ではサポートされません。

HP OpenVMS Graphical Configuration Manager によって行われるネットワーク通信はすべて，SSL (Secure Sockets Layer) を使用します。 GCM 管理データベースは暗号化されます。

GCM クライアントから， OpenVMS システム管理者は 1 つ以上の GCM サーバへのセキュアな接続を確立して， 次の機能を実行することができます。

- パーティション分割された AlphaServer の構成を表示する。
- 現在のハードウェア・プラットフォームのホットスワップ機能を利用する。
- パーティション分割された複数のインスタンスの間で分散コマンドを実行する。
- ソフト・パーティション分割されたインスタンス間でリソースを再割り当てする。
- リソース固有の特性を表示する。
- 1 つ以上のインスタンスをシャットダウンまたはリブートする。
- 追加の管理ツールを起動する。
- Galaxy 構成モデルを作成し，組み込む。
- オンライン・ドキュメントを表示する。

アソシエーション (association) とは， それぞれに GCM サーバがインストールされている OpenVMS インスタンスのグループであり，共通の GCM 管理データベースを共用します。 アソシエーションを定義することによって，システム管理者は幅広い配置方式の中から運用ニーズに適したものを使用することができます。 たとえば，大規模なシステムではハード・パーティションを作成して，部門ごとに使用することができます。これらのハード・パーティションの中にソフト・パーティションを作成して，個別のワーク・グループやユーザ・グループで使用することができます。 このような複雑な環境では，ハード・パーティションとソフト・パーティションのセットごとに個別のシステム管理者が存在する場合があります。また，すべてのパーティション，あるいはパーティションのサブセットの責任者となる総合管理者を定義することもできます。GCM では，特定のアソシエーションを定義して，特定のアクセス特権を個別のユーザに認可することができます。

最も単純な配置では，1 つの GCM サーバが単一のハード・パーティションの単一のソフト・パーティションで実行されます。管理データベース内のアソシエーション・レコードは，ハー

ド・パーティション (およびそのソフト・パーティション) を，個別のシステムとして管理される一意なエンティティとして識別します。

最も複雑な配置では，多くの異なるシステム (GS シリーズ，ES40，4100，8xxx) のそれぞれのハード・パーティション内のそれぞれのソフト・パーティションで GCM サーバが実行されます。管理データベースのアソシエーション・レコードは，管理対象となる複合エンティティを形成するシステムのグループを識別します。アソシエーションが複数のシステムとパーティションで構成されるとき，それぞれの GCM サーバはアソシエーション内の他の GCM サーバに対して自分自身を識別し，それによってセキュアな通信グリッドが確立され，高度な分散管理機能を行うことができます。

14.2 インストールの前提条件

このセクションでは，GCM をインストールして実行するためのソフトウェアおよびハードウェア要件を示します。

14.2.1 ソフトウェア要件

GCM サーバ:

- OpenVMS Alpha バージョン 7.2-1H1 以上
- HP TCP/IP Services for OpenVMS (HP OpenVMS に同梱)

GCM クライアント:

- OpenVMS
  - OpenVMS バージョン 7.2 以上
  - HP TCP/IP Services for OpenVMS (HP OpenVMS に同梱)
  - Java (JDK バージョン 1.2.2) ランタイム環境 (クライアントをインストールするとインストールされる)
- PC
  - Windows NT® または Windows® 2000
  - TCP/IP Services for OpenVMS
  - Java (JDK バージョン 1.2.2 以上) ランタイム環境

注意:

GCM クライアントは，現時点では，Java JDK バージョン 1.3 以上ではサポートされません。

14.2.2 ハードウェア要件

GCM の性能は，アソシエーションのサイズと構成によって左右されます。大きく複雑なアソシエーション (多くの GCM サーバを含んでいるもの) ほど，ネットワーク転送は低速になります。

GCM クライアントを実行するそれぞれのマシンに，800x600 ピクセル (SVGA) 以上のディスプレイが必要です。

GCM クライアントを実行するそれぞれの PC に，128 MB 以上のメモリが必要です。

14.3 インストール手順

このセクションでは，GCM のコンポーネントをインストールする方法を説明します。インストールは，次のようなステップで行います。

- 1 つの GCM サーバをインストールします (14.3.2 項「GCM サーバのインストール」を参照)。

- 1 つの GCM クライアントをインストールします (14.3.3 項「GCM クライアントのインストール」を参照)。
- GCM サーバのセットアップ・タスクを実行して，基本的な構成を確立します。administrator 特権を持つ初期ユーザの登録も含みます (14.3.4 項「GCM サーバのセットアップ」を参照)。
- GCM クライアントを (administrator 特権で) 実行して， アソシエーションの構成を終了します (14.3.4 項「GCM サーバのセットアップ」を参照)。
- GCM 管理データベースを送信します (14.3.4 項「GCM サーバのセットアップ」を参照)。
- 追加のユーザを登録します (14.3.4 項「GCM サーバのセットアップ」を参照)。

14.3.1 キット

OpenVMS GCM サーバおよびクライアント用の POLYCENTER Software Installation ユーティリティ・キットの最新版は，常に次の OpenVMS の Web サイトにあります。

http://www.hp.com/go/openvms/

[Products] をクリックして，OpenVMS クライアントとサーバ用のダウンロード可能なキットと，Windows クライアント用の zip ファイルを探します。

GCM は，OpenVMS Alpha Version 7.3–2 Layered Product CD-ROM にも入っています。 この CD-ROM は，オペレーティング・システムに同梱されています。

14.3.2 GCM サーバのインストール

GCM サーバは，OpenVMS バージョン 7.3-1 およびそれ以降のバージョンではプリインストールされています。以前のバージョンの OpenVMS (バージョン 7.2-1H1 以上) では， POLYCENTER Software Installation ユーティリティ・キットを使用して GCM サーバをインストールすることができます (14.3.1 項「キット」を参照)。

GCM サーバのインストールでは，次の操作が行われます。

- SYS$COMMON:[SYSEXE]GCM_SERVER.EXE イメージをインストール (またはアップデート) します。
- SYS$COMMON:[SYS$STARTUP]GCMSRV$STARTUP.COM ファイルをインストールします。
- SYS$COMMON:[SYS$CONFIG] ディレクトリを論理名 GCMSRV$DIR で定義します。
- 次のファイルをディレクトリ GCMSRV$DIR にインストールします。
  - GCM_RULESET.XML
  - GCM_CUSTOM.XML
  - GCM_BANNER.JPG
  - GCM_NOTICE.HTML
  - GCM$SETUP.EXE
  - GCM_CERT.PEM
  - GCM_PRIVKEY.PEM

例 14-1「GCM サーバのインストール例」に，GCM サーバのインストール例を示します。

例 14-1 GCM サーバのインストール例

```
$ PRODUCT INSTALL GCM_SERVER

The following product has been selected:
  CPQ AXPVMS GCM_SERVER V1.0              Layered Product

Do you want to continue?  [YES]

Configuration phase starting . . .

You will be asked to choose options, if any, for each selected product
and for any selected products that may be installed to satisfy software
dependency requirements.

CPQ AXPVMS GCM_SERVER V1.0:  Graphical Configuration Manager V1.0

     COPYRIGHT (C) 2002 -- All rights reserved

     Compaq Computer Corporation

     License and Product Authorization Key (PAK) Information

* This product does not have any configuration options.

     Copying GCM Release Notes to SYS$HELP

     Will install the GCM Server V1.0.

     GCM Startup File

Execution phase starting . . .

The following product will be installed to destination:
    CPQ AXPVMS GCM_SERVER V1.0        DISK$WFGLX5_X931:[VMS$COMMON.]

Portion done:  0%...10%...20%...30%...90%...100%

The following product has been installed:
    CPQ AXPVMS GCM_SERVER V1.0        Layered Product
$
```

例 14-2「GCM サーバ・ファイルのディレクトリ」に，GCM サーバがインストールされた後の現在のディレクトリのファイルを示します。

例 14-2 GCM サーバ・ファイルのディレクトリ

```
$ SET DEFAULT SYS$COMMON:[SYS$CONFIG]
$ DIRECTORY

Directory SYS$COMMON:[SYS$CONFIG]

GCM$SETUP.EXE;1   GCM_BANNER.JPG;1   GCM_CERT.PEM;1    GCM_CUSTOM.XML;1
GCM_NOTICE.HTML;1 GCM_PRIVKEY.PEM;1  GCM_RULESET.XML;1 GCM_SERVER.COM;1
```

14.3.3 GCM クライアントのインストール

14.3.3.1 項「OpenVMS GCM クライアント」では，OpenVMS GCM クライアントのインストール方法を示します。14.3.3.2 項「PC GCM クライアント」では，PC GCM クライアントのインストール方法を示します。

14.3.3.1 OpenVMS GCM クライアント

例 14-3「OpenVMS GCM クライアントのインストール」に，OpenVMS GCM クライアントのインストール例を示します。

例 14-3 OpenVMS GCM クライアントのインストール

```
$ PRODUCT INSTALL GCM_CLIENT

The following product has been selected:
  CPQ AXPVMS GCM_CLIENT V1.0              Layered Product

Do you want to continue?  [YES]

Configuration phase starting . . .

You will be asked to choose options, if any, for each selected product
and for any selected products that may be installed to satisfy software
dependency requirements.

CPQ AXPVMS GCM_CLIENT V1.0:  Graphical Configuration Manager V1.0 Client

     COPYRIGHT (C) 2002 --- All rights reserved

     Compaq Computer Corporation

     License and Product Authorization Key (PAK) Information

* This product does not have any configuration options.

     Copying GCM Release Notes to SYS$HELP

     Will install the GCM Java Client V1.0.

     Will install a private copy of the Java JRE V1.2.2-3.

     Will install a private copy of the Java Fast VM V1.2.2-1

Execution phase starting . . .

The following product will be installed to destination:
    CPQ AXPVMS GCM_CLIENT V1.0        DISK$WFGLX5_X931:[VMS$COMMON.]

Portion done:  0%...10%...20%...30%...90%...100%

The following product has been installed:
    CPQ AXPVMS GCM_CLIENT V1.0        Layered Product
$
```

例 14-4「GCM クライアント・ファイルのディレクトリ」に，OpenVMS GCM クライアントがインストールされた現在のディレクトリのファイルを示します。OpenVMS GCM クライアントをインストールすると，JRE.DIR ディレクトリに Java ランタイム環境がインストールされることに注目してください。

例 14-4 GCM クライアント・ファイルのディレクトリ

```
$ SET DEFAULT SYS$COMMON:[GCM_CLIENT]
$ DIRECTORY

Directory SYS$COMMON:[GCM_CLIENT]

BIN.DIR;1  GCM_CERT.CRT;1  IMAGES.DIR         JRE122.DIR;1
LIB.DIR;1  README.TXT;1;1  RUN_GCMCLIENT.COM;1
```

14.3.3.2 PC GCM クライアント

PC GCM クライアントをインストールするには，まず，GCM クライアントの zip ファイルをコピーします。 このファイルは，次の OpenVMS ソフトウェアの Web サイトにあります。

http://www.hp.com/go/openvms/

次に zip ユーティリティを使用して GCM クライアントをインストールします。

GCM は，OpenVMS Alpha Version 7.3–2 Layered Product CD-ROM にも入っています。

インストール・プロセスでは，GCM クライアントのインストール先の Windows ディレクトリを選択することができます。 GCM クライアント・セッションで毎回同じ環境設定を使用するには，インストール時に選択したディレクトリから GCM クライアントを起動する必要があります。 このためには，「GCM Client」アイコンをクリックします。

14.3.4 GCM サーバのセットアップ

GCM サーバをセットアップするには， まず，デフォルトを SYS$COMMON:[SYS$CONFIG] ディレクトリに設定した後， GCM$SETUP.EXE を実行して次のタスクを行います。

- アソシエーションの初期構成を定義します。
- 初期ユーザを登録します (ユーザ名，パスワード，および特権)。
- 今すぐ GCM サーバを起動するかどうかを選びます。
- システム起動時に自動的に GCM サーバを起動するかどうかを決めます。

GCM サーバのセットアップ中に， 追加の GCM ユーザを登録するかどうかも選ぶことができます。

GCM$SETUP.EXE を実行すると， GCM 管理データベース・ファイル SYS$COMMON:[SYS$CONFIG] GCM_ADMIN.EDB が生成され，暗号化されます。 GCM サーバの起動を選択すると， SYS$STARTUP:GCMSRV$STARTUP.COM プロシージャが実行されて，次のような操作が実行されます。

- GCMSRV$DIR:GCM_STARTUP.COM が作成されて，実行されます。 これにより，GCM サーバが独立プロセスとして起動され，ログ・ファイルは GCMSRV$DIR:GCM_SERVER.LOG が使用されます。
- プロセス名前 GCM_SERVER というプロセス名で GCM サーバを起動します。

注意:

GCM サーバは，サーバ障害後は自動的に再起動します。 GCM サーバがハングした場合はタイムアウトになり，再起動します。 適切な特権を持つ GCM クライアントから RESTART コマンドを入力することもできます。

初めて GCM サーバが再起動したときのプロセス名は GCM_SERVER01 です。 その後，GCM サーバが再起動するたびに，プロセス名は 1 ずつ増分されます。 プロセス名が GCM_SERVER99 になると， GCM サーバが再起動ループに入るのを防止するために， その後の再起動は禁止されます。

システム論理名 GCMSRV$ABORT を定義することによって， 自動再起動の試みを防ぐことができます。再起動の試みの原因を取り除いた後，この論理名の割り当てを解除して，GCM サーバの自動再起動を有効にしておく必要があります。

例 14-5 「GCM サーバのセットアップ例」 に， 初期の GCM サーバ構成のセットアップ例を示します。

例 14-5 GCM サーバのセットアップ例

```
$ SET DEFAULT SYS$CONFIG
$ DIRECTORY

Directory SYS$COMMON:[SYS$CONFIG]

GCM$SETUP.EXE;1     GCM_BANNER.JPG;1    GCM_CERT.PEM;1      GCM_CUSTOM.XML;1
GCM_NOTICE.HTML;1   GCM_PRIVKEY.PEM;1   GCM_RULESET.XML;1

Total of 7 files.
$ RUN GCM$SETUP

OpenVMS GCM-Server Setup Utility
Copyright 2002, Hewlett Packard Corporation

This utility initializes the GCM Administrative Database:
SYS$COMMON:[SYS$CONFIG]GCM_ADMIN.EDB

If you are performing an initial GCM-Server installation that will
create an Association of more than a single server instance, you must
perform the following tasks to assure proper server synchronization:

1) Create the new local database using this utility.
2) Copy the database to all other GCM-Server instances in your
   Association.
3) Start each GCM-Server.

You can start servers via $@SYS$STARTUP:GCMSRV$STARTUP.COM or by
invoking this utility on each system.  When an existing database is
found, this utility will offer additional server startup options.

Continue (Y/N) {Y}?

Do you prefer full text assistance (Y/N) {N}? y

SERVER DISCOVERY TCP/IP PORT NUMBER:

Use the default Port Number 4100 (Y/N) {Y}? y

CONNECTION BOUNDS:

Use the default concurrent CLIENT limit of 4 (Y/N) {Y}? y

Use the default concurrent SERVER limit of 8 (Y/N) {Y}? y

CONNECTION SECURITY:

WARNING - DISABLING SECURITY COULD POTENTIALLY COMPROMISE THE
INTEGRITY OF YOUR SYSTEM AND IS NOT RECOMMENDED BY HEWLETT PACKARD.

Use SSL Client-Server Security (Y/N) {Y}? y

%SECURITY - WILL BE ENABLED.

SERVER ASSOCIATION RECORD:

Enter the Association Name (use no quotes): GCM Test Association

SYSTEM RECORDS:

Enter a SYSTEM Name (ex: star.zko.hp.com) (RETURN when done):
      wfglx4.zko.hp.com

Enter a System IP Address (0 to use DNS): 16.32.112.16

Enter a System Number (range 0-7): 0

Enter a HARD PARTITION Number (range 0-7): 0

Enter a SOFT PARTITION Number (range 0-7): 0
```

```
%Define additional system records as needed...

Enter a SYSTEM Name (ex: star.zko.hp.com) (RETURN when done):
        wfglx5.zko.hp.com

Enter a System IP Address (0 to use DNS): 16.32.112.17

Enter a System Number (range 0-7): 0

Enter a HARD PARTITION Number (range 0-7): 0

Enter a SOFT PARTITION Number (range 0-7): 1

%Define additional system records as needed...

Enter a SYSTEM Name (ex: star.zko.hp.com) (RETURN when done):

CLIENT AUTHORIZATION RECORDS:

Enter Client's Full Name or Title (RETURN when done): First Last

Enter Client's Username: First

Enter initial Password for First: Last

Enter EMAIL Address for First: First.Last@hp.com

Give First CONFIG Privilege (Y/N) {Y}? y

COMMAND privilege allows a user to issue DCL commands.

Give First COMMAND Privilege (Y/N) {Y}? y

Give First USER-COMMAND Privilege (Y/N) {Y}? y

Give First POWER Privilege (Y/N) {Y}? y

Give First ADMIN Privilege (Y/N) {Y}? y

Enable First account now (Y/N) {Y}? y

%Define additional client records as needed...

Enter Client's Full Name or Title (RETURN when done):

SERVER STARTUP OPTIONS:

Do you want the local GCM-SERVER to start on System Boot (Y/N) {Y}? y

Do you want to start the local GCM-SERVER now (Y/N) {Y}? y


********************** POST SETUP TASKS *************************

 This completes the GCM Admin Database initialization.

 IMPORTANT:

 If you are using multiple GCM-Servers, copy the newly created GCM
 Admin Database file: SYS$COMMON:[SYS$CONFIG]GCM_ADMIN.EDB, to the
 same location on each system you defined in the Association, then
 start or restart each server via $ @SYS$STARTUP:GCMSRV$STARTUP.COM

 If the database has been properly defined, each server will detect
 the presence of the other servers and form the specified Association.
 If the GCM_SERVER process fails to start, examine the server logfile
 SYS$COMMON:[SYS$CONFIG]GCM_SERVER.LOG.

 When the server has started, you may use the GCM-Client to establish
 a connection and further tune the installation.
```

```
 For maximum security, you may wish to protect or remove this utility.

$
```

各エントリを説明する完全なプロンプト・テキストを含んだ GCM サーバのセットアップ例については， 14.12 項「冗長な GCM サーバ・セットアップの例」 を参照してください。

GCM 管理データベース

GCM$SETUP ユーティリティは，システム構成，ユーザ識別情報， および起動方法に関する詳細の入力を求めます。

セキュリティを強化するには，GCM セットアップ・ユーティリティを実行した後， 名前の変更，削除，またはその他の方法でこのユーティリティを保護してください。 セットアップ・ユーティリティが再び必要になるのは， 初期データベースを再構築する必要がある場合だけです。

初期セットアップ時に，GCM サーバを実行するシステムを少なくとも 1 つ定義し (通常は，ログインしているシステム)， また，administrator 特権を持つユーザを登録する必要があります。 このユーザは，後で GCM クライアントを実行して GCM サーバに接続し， 詳細な構成と調整を行います。GCM サーバのセットアップ中に，アソシエーションに追加のインスタンスを定義することができ，また，個別のアクセス特権を持つ追加の GCM ユーザを登録することができます。 また，これらのタスクは，GCM クライアントからいつでも行うことができます。

GCM セットアップ・ユーティリティは， SYS$COMMON:[SYS$CONFIG]GCM_ADMIN.EDB ファイルを作成します。 これは，初期の GCM 管理データベースです。これは， アソシエーションとそのユーザに関する情報を含む暗号化されたデータベースです。 通常の操作では，このデータベースを解釈できるのは GCM サーバだけです。 (14.9 項「GCM サーバのトラブルシューティング」 を参照。) 複数のインスタンスを持つアソシエーションを構成した場合には， それらのインスタンス上で GCM サーバを起動する前に， GCM 管理データベースを追加インスタンスのそれぞれの GCM サーバ・ディレクトリに手動でコピーする必要があります。 データベース・ファイルをコピーしないと， アソシエーションが正しく作成されなかったり， 初期の GCM 管理データベースが上書きされることがあります。

新しいインスタンスの追加やインスタンスの削除など，アソシエーションの構成の変更は主要な変更です。アソシエーションに主要な変更を加えたときには必ず，GCM サーバを起動する前に，そのインスタンス上の GCM サーバ・ディレクトリに GCM 管理データベース・ファイルを手動でコピーする必要があります。

アクティブなアソシエーション内のデータベースにマイナーな変更を加えた場合は，すべての GCM サーバに自動的に伝播されます。 GCM サーバを再起動する必要はありません。 マイナーな変更としては，GCM クライアント・レコードの追加と削除や GCM サーバ・パラメータの調整などが含まれます。

14.4 GCM サーバの起動

GCM サーバは自動 (14.4.1 項「GCM サーバの自動起動」) または手動 (14.4.2 項「GCM サーバの手動起動」) で起動することができます。

1 つのインスタンスでは 1 つの GCM サーバだけを実行してください。

14.4.1 GCM サーバの自動起動

サーバのセットアップ時に GCM サーバの自動起動を選択した場合 (14.3.4 項「GCM サーバのセットアップ」 参照)，セットアップ・ユーティリティは次のコマンドを実行します。

```
$ MC SYSMAN STARTUP ADD FILE GCMSRV$STARTUP.COM
```

このコマンドはいつでも手動で入力することができます。

GCMSRV$STARTUP.COM プロシージャは，次のようなタスクを実行します。

- 必要なシステム論理名を定義します。
- 小さなワーカー・プロシージャ SYS$COMMON:[SYS$CONFIG]GCM_SERVER.COM を作成します。
- GCM サーバ・プロセスを独立プロセスとして実行します。

自動起動をオフにするには，次のコマンドを入力します。

```
$ MC SYSMAN STARTUP REMOVE FILE GCMSRV$STARTUP.COM
```

14.4.2 GCM サーバの手動起動

次のコマンドを入力することによって，いつでもGCMサーバを手動で起動することができます。

```
$ @SYS$STARTUP:GCMSRV$STARTUP
```

14.5 インストール後の管理タスク

管理者は，特権 GCM クライアントから次のタスクを行います。

- 他の GCM ユーザを登録します (14.5.1 項「GCM ユーザの登録とサブスクリプション・テンプレートの定義」を参照)。
- ライブラリにドキュメントを投稿します (オプション) (14.5.2 項「ライブラリの管理 」を参照)。
- GCM_BANNER.JPG ログイン・バナーを定義します (オプション) (14.5.3 項「カスタム・バナー」を参照)。
- システム通知 GCM_NOTICE.HTML ファイルを投稿します (オプション) (14.5.4 項「システム通知」を参照)。
- カスタムのシステム・コマンドとユーザ・レベル・コマンドを定義します (14.5.5 項「コマンドのカスタマイズ」を参照)。

これらのタスクの詳細については，以下のセクションとGCMクライアントのヘルプを参照してください。

14.5.1 GCM ユーザの登録とサブスクリプション・テンプレートの定義

GCM サーバのセットアップ手順において， または GCM クライアントからいつでも (admin 特権で)， GCM ユーザの定義，登録，および有効化を行うことができます。 典型的なインストールでは， セットアップ・ユーティリティからシステムの主要なユーザを定義します。 その他のユーザは，サブスクリプション・プロセスと電子メール要求を使用して， GCM サーバ・アソシエーションへのアクセスを要求することができます。

インストール後のセットアップの一部として， GCM サーバは初めて起動されたときにサブスクリプション・テンプレート・ファイル SYS$COMMON:[SYS$CONFIG]GCM_TEMPLATE.XML を作成します。 このファイルは，特定のサイトに応じたサブスクリプション・フォームを作成するテンプレートを含んでいます。 このテンプレートをカスタマイズするには， 管理者は GCM クライアントから GCM サーバに接続して， 「File」メニューの「Edit Subscription Template」を選択する必要があります。

デフォルトのサブスクリプション・テンプレートが表示されたら，必要に応じてフォームに入力して「OK」を押すと， 公式テンプレートがすべてのアクティブな GCM サーバに返されます。 (GCM サーバがまだ実行されていない場合は， このファイルを構成内の他のインスタンスに手動でコピーすることもできます。) サブスクリプション・フォームには，潜在的ユーザがユーザ名，パスワード， およびアクセス・レベルを要求するためのフィールドがあります。 管理者は，ユーザに表示する認可を選ぶことができます。 たとえば，システム管理者アカウントを 1 つだけにしておきたい場合は， ユーザが Admin 特権を要求できないようにすることができます。

注意:

GCM ユーザ登録レコード (ユーザ名，パスワードなど) は， OpenVMS の認可メカニズムから完全に独立しています。

サブスクリプション・フォームが初期化されたら，ユーザは GCM 対応のインスタンス (GCM サーバを実行しているインスタンス) を見つけて， サブスクリプション・フォームを要求することができます。 管理者がサブスクリプション・テンプレートをカスタマイズするまでは，ユーザはサブスクリプション・フォームを要求できません。 ユーザが入力したサブスクリプション・フォームは GCM サーバに返され， 新しいサブスクリプション要求を通知する電子メールが管理者に送られます。要求を承認するには，管理者は GCM クライアントからアソシエーションに接続して， GCM 管理データベースを編集し，要求を承認または拒否するオプションを選択します。オプションとして，管理者は「Email」ボタンをクリックして， 要求のステータスについてユーザに助言する応答メールを表示することができます。 サブスクリプション要求が承認されて，アカウントが有効になると，更新された GCM 管理データベースが自動的に他の GCM サーバに送信され， 要求した GCM ユーザはアソシエーション内の GCM サーバへのアクセスを許されます。

GCM ユーザのアクセス権を無効にするには， 管理者はクライアント・レコードを削除するか，単にアカウントを無効にします。 現在接続しているクライアントは，ただちに切断されます。

GCM ユーザの特権

ユーザを登録するときに，次のような特権を与えることができます。

特権	説明
Config	ユーザはアソシエーションの構成を変更することができる。
Command	ユーザは自分用のコマンドを定義することができる。
User Command	ユーザはすべてのクライアント用のコマンドを定義することができる。
Admin	ユーザは GCM 管理データベース (GCM$ADMIN.EDB) を編集することができる。
Power	ユーザはシステムとコンポーネントの電源投入および電源切断を行うことができる。

14.5.2 ライブラリの管理

GCM には分散文書検索システムが含まれています。このシステムによって， 管理者は個別の GCM サーバ上の指定したディレクトリにドキュメントを投稿することができ，ユーザはそれらを検索し，表示することができます。 (「Library」ダイアログ・ボックスを呼び出すには，「File」メニューの「Edit Admin Database」を選択します。) さらに，GCM サーバ・システム上で実行されるプログラムとプロシージャは， ユーザが検索・表示するための出力ファイルを生成することができます。

ドキュメントの検索にはワイルドカード文字を使用することができます。たとえば，リモート GCM サーバ・コマンド・プロシージャを実行して性能分析を行い， Excel スプレッドシート出力ファイルを生成することができます。GCM を使用すると， このプロシージャをアソシエーション内の複数のシステムで同時に起動することができます。それぞれのシステムが生成する結果ファイルのファイル名には，ノード名やタイムスタンプなどの一意な識別名が含まれています。 たとえば，CFG_*_LOG.CSV など，ワイルドカードでドキュメントを指定することができます。このように指定すると，それぞれのシステムから一致するファイルが検索されます。一致したファイルは，検索に使用可能なドキュメントとして GCM クライアントのライブラリにリストされます。 完全な OpenVMS ディレクトリの指定もできます。

ユーザが GCM サーバ・ライブラリを開いてドキュメントを検索するときには常に， そのドキュメントのコピーがユーザの GCM クライアント・ディレクトリに書き込まれます。ユーザが GCM クライアント・セッションを終了した後も，これらのファイルをエディタやブラウザで表示することができます。 ただし，ファイルに加えられた変更は，GCM サーバ上のライブ

ラリにコピーされません。 これらのファイルが不要になったときには， ユーザは手動で削除しなければなりません。

14.5.3 カスタム・バナー

独自の JPEG ファイルを SYS$COMMON:[SYS$CONFIG]GCM_BANNER.JPG にコピーすることによって， 会社または組織のカスタム・バナー・イメージを提供することができます。 ダウンロードに時間がかかり過ぎないように，このファイルは適切なサイズでなければなりません。 このファイルは，接続時に GCM クライアントに表示されます。

14.5.4 システム通知

SYS$COMMON:[SYS$CONFIG]GCM_NOTICE.HTML ファイルを編集することによって， GCM クライアント用の一般的なシステム通知を記入することができます。 このファイルは単純なハイパーテキスト文書であり， CM クライアントが GCM サーバに接続したときに表示されるメッセージを含めることができます。 このファイルの編集は単純にしておくべきです。 リンクを埋め込むことはできますが，スクリプト，イメージ， サウンドなどを埋め込むことはできません。 GCM クライアントの組み込みブラウザ (Java デフォルト・ブラウザ) は， 最新のブラウザほど高度な機能を備えていず，汎用目的のものではありません。

14.5.5 コマンドのカスタマイズ

GCM コマンドをカスタマイズするには 2 つの方法があります。 ユーザ・コマンド特権を持つユーザは，カスタム DCL コマンドを定義・保存するためのメニュー項目にアクセスできます。 これらのコマンドは，OpenVMS と DCL によってサポートされている関数であれば，どんな関数でも実行することができます。GCM サーバは， 現時点では，対話型コマンドやコマンド・プロシージャをサポートしていません。

注意:

ユーザ・コマンド特権を与えるときには， すべてのコマンドが GCM サーバによってシステム・コンテキストにおいて実行されることを覚えておいてください。 GCM は，システムの性能や運用に有害なコマンドの実行を防止しようとはしません。

拡張ルールセット定義によってカスタム・コマンドを作成することもできます。 それぞれの GCM サーバ上のオプション・ファイル (SYS$COMMON:[SYS$CONFIG]GCM_CUSTOM.XML) は， XML 形式の単純なメニュー定義文を含みます。 GCM サーバの起動時にこのファイルが存在した場合， コマンド特権を持つすべてのユーザについて， これら追加のメニュー・コマンドが GCM クライアントに表示されます。これは，システム管理者がユーザのためにインスタンス固有のコマンドを定義する手段となります。 キットにサンプル・ファイルが含まれていて， 追加のカスタム・コマンドの定義に関する説明が含まれています。 この機能を使用しない場合は，カスタム・ルールセット・ファイルを削除しても構いません。 カスタム・コマンドの使用の詳細については，14.7.1 項「コマンドの定義と使用」 を参照してください。

14.6 アソシエーションの構成

GCM が正常に動作するためには，システムの，そしてハード・パーティションとソフト・パーティションの正しい値を指定することが重要です。 値が正しくないと，コマンド・ルーティングが失敗します。 GCM のインストールとアソシエーションの構成の前に， ワークシート (作業計画書) でアソシエーションを定義しておくとよいでしょう。

表 14-1 「単純なアソシエーションのワークシート例」 は，2 つのハード・パーティションのそれぞれに 4 つのソフト・パーティションを含んでいるシステムのアソシエーションを定義する方法の例です。 [1]

1. 値がゼロの場合，DNS 名前変換では現在の IP アドレスが検索されます。

表 14-1 単純なアソシエーションのワークシート例

完全修飾されたシステム名	IP アドレス	システム	ハード・パーティション	ソフト・パーティション
sysnam1.zko.dec.com	16.32.112.1	0	0	0
sysnam2.zko.dec.com	16.32.112.2	0	0	1
sysnam3.zko.dec.com	16.32.112.3	0	0	2
sysnam4.zko.dec.com	16.32.112.4	0	0	3
sysnam5.zko.dec.com	16.32.112.5	0	1	0
sysnam6.zko.dec.com	16.32.112.6	0	1	1
sysnam7.zko.dec.com	16.32.112.7	0	1	2
sysnam8.zko.dec.com	16.32.112.8	0	1	3

14.6.1 考慮事項

アソシエーションは，単一システム， 単一システム内のハード・パーティションとソフト・パーティションの組み合わせ，またはハード・パーティションとソフトパーティションの組み合わせを含む複数のシステムです。 一般に，GCM は，そのシステムに存在するすべてのソフト・パーティションを含めて，単一システムで使用されます。 GCM アソシエーションを複数のシステム (以前の AlphaServer システムも含めて) に拡張することもできますが，大きなアソシエーションを作成した場合の性能への影響に留意してください。

構成を変更すると，GCM によるネットワーク・トラフィックが急増します。 アソシエーションが多くのシステムとパーティションにまたがる場合， データ量が過剰になることがあります。 弊社は GCM の動作の改善を続けていますが， アソシエーションの定義方法によってはアソシエーションが管理不能になることがあります。ネットワークの速度や負荷， GCM クライアントのシステム・リソースなど， さまざまな要素が性能に影響を与えるので， アソシエーションのサイズに制限を設定するのは困難です。 GCMの最初のリリースの設計目標は，最大 8 つのパーティションです。 一般的なインストールでは，パーティションの数はこれよりはるかに少ないです。

14.6.2 アソシエーションへのシステムの追加

システム とは，物理的なマシン本体のことです。 アソシエーションにシステムを追加するには，GCM セットアップ・ユーティリティを使用して追加するか (少なくとも 1 つのローカル・システムを定義する必要があります。14.3.4 項「GCM サーバのセットアップ」を参照)，またはいつでもGCMクライアントを使用して追加することができます。 アソシエーションに新しいシステムを追加するときには，識別の詳細を指定する必要があり，また，必要に応じて GCM サーバの限界を調整する必要があります。

それぞれのシステムには一意なシステム番号を割り当てなければなりません。 たとえば，GS320 では，1 つのシステムに最大 8 つのパーティションを設定できます。 システム番号はゼロから連続的に割り当てられます。 システム番号ごとに個別の構成表示が作成されます。

それぞれのシステムのそれぞれのハード・パーティションにハード・パーティション番号を定義する必要があります。 これは，実際のハード・パーティション番号でなければなりません。 たとえば，GS320 を 4 つのハード・パーティションに分割した場合は， 0 から 3 までのパーティション番号が割り当てられます。

それぞれのハード・パーティション内のそれぞれのソフト・パーティションにソフト・パーティション番号を定義する必要があります。 これは，実際のソフト・パーティション番号でなければなりません。 たとえば，ハード・パーティション番号 0 をソフト・パーティション 0 と 1 に分割した場合は， それぞれのシステムに適切なパーティション番号を指定しなければなりません。 表 14-1 「単純なアソシエーションのワークシート例」 は， このようにして構成されたシステムを示しています。

すべてのパーティションがシステム 0 にあることに注目してください。 独自のパーティション・セットを持つ別のシステムを追加してアソシエーションを拡張した場合，それらのパーティションはシステム 1 になります。

注意:

これらの値を正しく設定することが重要です。適切な値を指定しなかった場合，GCM は目的のアソシエーションを形成できなかったり，アソシエーションを通じてコマンド・トラフィックを正しくルーティングできません。CONFIGURE GALAXY コマンドでネイティブの Graphical Configuration Utility (GCU) を起動して，GCM の適切な識別子を調べなければならない場合があります。

GCM サーバは， サポートする並列 GCM クライアントおよび GCM サーバの上限数を保持します。 デフォルトは，4 つの並列 GCM クライアントと 8 つの並列 GCM サーバです。 これらの限界に達すると， 追加の GCM クライアントまたは GCM サーバ接続要求は拒否されます。 特権 GCM クライアントから GCM 管理データベースを編集することによって， これらの値を変更することができます。

変更後は，変更した GCM 管理データベース・ファイル GCM_ADMIN.EDB をアソシエーションに追加されたすべてのインスタンスに手動で伝播してください。 GCM サーバはすでに実行したことがあるが，このアソシエーションでは初めて実行するという場合には，すべての GCM サーバを停止して， GCM 管理データベース・ファイルを手動でコピーし， すべての GCM サーバを再起動します。アソシエーションがセットアップされ，正常に機能するようになったら，必要以上に変更しないでください。 一度，安定性が達成されると，GCM はその後も安定して動作するはずです。

14.7 GCM のカスタマイズ

次のような方法で GCM をカスタマイズすることができます。

- GCM クライアント・アプリケーションのジオメトリをカスタマイズします。

  これには，ウィンドウの色，サイズ，および配置が含まれます。 これを行うには，GCM クライアントのメニュー機能を使用します。 詳しくは，GCM クライアントのヘルプを参照してください。

- 接続属性をカスタマイズします。

  これには，IP ポートの割り当て，接続時に通知またはバナーを検索するかどうか，さまざまな制限と診断属性が含まれます。 これらの属性はすべて，GCM クライアントから変更することができます。 IP ポートの割り当てを変更した場合は，GCM サーバを再起動する必要があります。

- カスタム・コマンドを定義します。

  詳しくは，14.7.1 項「コマンドの定義と使用」 を参照してください。

14.7.1 コマンドの定義と使用

GCM は OpenVMS DCL コマンドおよびコマンド・プロシージャの分散実行をサポートします。ただし，現時点では，GCM は対話型コマンドおよびプロシージャをサポートしていません。 GCM を使用して実行される可能性があるコマンドとして DCL コマンドがありますが，DCL コマンドには次のような特性があります。

- すぐに実行されます。
- 実行時に出力を生成します。
- すぐに戻ります。

DCL コマンドの多くはユーザからの応答を必要とし，応答がない場合はすぐには戻りません。このようなコマンドは，GCM での使用に適していません。 たとえば，RUN コマンドは常にすぐ戻るわけではありません。 GCM サーバはそれぞれのコマンド要求をサブプロセスで実行しますが， ユーザに応答を返さず，また， プログラムが終了しなければサブプロセスを終了

しません。したがって，サブプロセスを停止するには，SHOW SYSTEM コマンドを入力して，該当するサブプロセスを調べてから手動で停止しなければなりません。

すべての GCM サーバ・コマンドには GCM_nodename_CMDnnnn という形式のサブプロセス名が割り当てられます。nnnn は，コマンドの連続番号です。一般に，これらのサブプロセスは完了まで実行して，GCM サーバに AST を送ります。 それによって GCM サーバは，サブプロセスと同じ名前の一時ログ・ファイルから関連する出力を見つけることができます。結果ファイルを返した後，サブプロセスと関連ファイルは削除されます。サブプロセスが，完了しないコマンドや入力を待つコマンドを実行している場合， GCM サーバにコマンド完了通知が送られないので，GCM サーバは結果を返しません。 このため，GCM サーバは他の処理を行えなくなり， 予期しないコマンド動作につながることがあります。

GCM の将来のリリースでは，追加のコマンド機能が提供される予定です。しかし，サポートできない種類のコマンド機能は常に存在するでしょう。グラフィカル・インタフェースや複雑な対話を伴なうアプリケーションの起動は，GCM の使用目的の範囲外です。多くの場合，単純なコマンド・プロシージャ・ラッパーを作成して，アプリケーションがそれ自身の表示のリダイレクトを管理できるようにすることが可能です。

14.8 GCM サーバのログ・ファイル

通常の GCM サーバ実行の一部として，いくつかのログ・ファイルが生成されます。これらのログ・ファイルのなかには，GCM サーバのトラブルシューティングに役立つものがあります (14.9 項「GCM サーバのトラブルシューティング」 参照)。 システム・ディスクがクラスタ共通のシステム・ディスクである場合に個々のシステム・ルートからログ・ファイルを分離する手段として， ログ・ファイルのファイル名の一部としてシステム・ノード名が含まれている場合があります。

次のようなログ・ファイルが生成されます。

- 実行時ログ・ファイル

  GCM サーバ・プロセスが開始されると，SYS$COMMON:[SYS$CONFIG]GCM_nodename_SERVER.LOG ログ・ファイルが作成されます。 このログ・ファイルには，GCM サーバ・フォールトのトラブルシューティングに役立つ詳細が含まれていることがあります。

- コマンド・ログ・ファイル

  GCM サーバ・コマンドが実行されるたびに，追加のログ・ファイルが作成されます。 これらのログ・ファイルの名前の形式は，次のとおりです。

  GCM_nodename_CMDsequence_number.LOG

  コマンド実行が成功すると，関連するログ・ファイルは削除されます。 まれに，GCM サーバ・フォールトのために， これらのログ・ファイルの 1 つが削除されずに残ることがあります。 残ったログ・ファイルは，GCM サーバの起動時に削除されます。

- 接続ログ・ファイル

  実行時に， 接続ログ・ファイルを生成するように GCM サーバを設定することができます。 このオプション・ファイル SYS$COMMON:[SYS$CONFIG]GCM_nodename_CONNECT.LOG は，成功および失敗した GCM クライアント接続のタイムスタンプ付き履歴を含んでいます。

  実行時に，GCM サーバ・イベント・ログを生成するように GCM サーバを設定することができます。 このオプション・ファイル SYS$COMMON:[SYS$CONFIG]GCM_nodename_EVENT.LOG は，GCM サーバのトラブルシューティングに役立つさまざまな情報を含んでいます。 このイベント・ログの内容は，GCM サーバの診断フラグによって変わります。 イベント・ログのエントリは，過度に大きいファイルが生成されるのを避けるために無効にされるのが一般的です。

14.9 GCM サーバのトラブルシューティング

通常の起動手順で起動したときには，GCM サーバは自動再起動動作をサポートするモードで実行されます。 通常モードで実行された場合，GCM サーバは実行時ログ・ファイルを生成しますが， GCM サーバ問題の解決に役立つ追加情報はほとんど記録されません。

以下のセクションには， GCM サーバ問題のトラブルシューティングに役立つ情報があります。

14.9.1 診断情報の取得

ときには， 何らかの診断が行われるモードで GCM サーバを実行すると有益なことがあります。 診断が生成する手順追跡情報は，GCM サーバ・イベント・ログに書き込まれるか (有効な場合)，画面に出力されます (有効な場合)。 一般に，診断出力は画面に出力されますが， 弊社のサポート担当者に転送する必要がある場合は，イベント・ログ出力は手助けになり，便利です。

GCM サーバから診断画面出力を得るには，独立プロセスとしてではなく， 対話方式で GCM サーバを実行する必要があります。 このためには，GCM サーバが実行中であれば停止して，DECterm ウィンドウから次のコマンドを入力して再起動します。

```
$ RUN SYS$SYSTEM:GCM_SERVER.EXE
```

GCM サーバが実行状態になったら，GCM クライアントから管理特権で接続して， GCM 管理データベースを次のように編集します。

1. 「File」メニューから「Edit Admin Database」を選択します。
2. 「Server Init」ページを選択します。
3. 「Diagnostic」テキスト・フィールドを探して， 目的の出力を選択するためのボタンを探します。
4. 「Screen output」ボタンをクリックし， 次の表に従って適切な診断コードを入力します。一度に複数のフラグを設定することができます。

コード (10 進)	コード (16 進)	機能	アクション
0	0	DIAGNOSTIC_DISABLE	診断の無効化。
1	1	HEARTBEAT_TRACE	サーバ切断。
2	2	TRANSACTION_TRACE	一般的なトラブルシューティング。
4	4	XML_TRACE	一般的なトラブルシューティング。
8	8	LOCK_TRACE	サーバのハング。
16	10	COMMAND_TRACE	実行の無効化とパケットのダンプ。
32	20	CRYPTO_DISABLE	GCM 管理データベースのトラブルシューティング。

注意:

デフォルトでは，GCM クライアント/サーバ通信でセキュリティ (SSL) を使用しているかどうかに関係なく，GCM 管理データベースは常に暗号化されます。まれに，GCM 管理データベース・ファイルの暗号化を無効にしたい場合があります。 CRYPTO_DISABLE フラグを設定すると， GCM サーバは GCM_ADMIN.EDB ファイルの暗号化と更新を行わなくなります。 代わりに，平文の ASCII XML を GCM_ADMIN.DAT に出力して， GCM_ADMIN.DAT からの起動時に入力を受け入れます。 このため，GCM 管理データベース内で XML 構造を直接編集して，GCM クライアントとサーバによって行われた変更を見直すことができます。

注意:

トラブルシューティングのとき以外は，CRYPTO_DISABLE フラグが設定された状態で GCM サーバを実行しないでください。 このフラグが設定された状態で GCM サーバを実行すると，GCM ユーザ登録レコードを単純な暗号化されていない ASCII 形式で公開することになるからです。

GCM サーバを実用に戻す前に，必ず診断を無効にしてください。ロギングを無効にしなかった場合，関連する GCM サーバ・ログ・ファイルが非常に大きくなり，GCM サーバの性能が低下することがあります。

14.9.2 潜在的な問題領域

ハートビートは，高負荷システムでは潜在的な問題領域です。 GCM サーバが他のインスタンスの GCM クライアントと GCM サーバの存在の有無を検出できるように，定期的なハートビート・トランザクションが発行されます。 GCM サーバは，ハートビート・プロセスを最適化し， 使用率に基づいて時間間隔を変えることができ， 一方向のハートビートしか必要としません。 ときには，GCM サーバまたはネットワークのアクティビティによってハートビートの発行が遅延することがあります。GCM サーバは，ハートビートを何回か受信しなくても切断とみなさないように設計されています。

GCM クライアントを使用して GCM 管理データベースを編集することによって， 特定のニーズに合わせてハートビートの値を調整することができます。 Client Pulse，Server Pulse，および Flat Line という 3 つの値が適用されます。 Pulse の値は，ハートビート・トランザクションの時間間隔 (秒数) です。 Flat Line の値は， 何回ハートビートが受信されなかったときに切断とみなされるかという回数です。 これらは，過度のトラフィックをもたらすことなく良好な動作をもたらす値に事前設定されています。特別な状況下で，セッションが切断されていることに気づいた場合， これらの値を変更することができます。これらの値は， コマンドとイベントの処理に関して GCM の全体的な応答性には影響を与えません。

14.9.3 タイムアウトの検出

GCM クライアントには， タイムアウト検出を完全にオフにするデバッグ・フックが含まれています。 デバッグ・モードに入るには，Ctrl+Alt+D+B を押します。「TEST」メニュー項目は，このモードがアクティブであることを示します。 このモードでは，低速な GCM サーバ (デバッグ中のものなど) から応答が得られるまでに時間がかかっても，GCM クライアントは常に接続状態を保ちます。

14.10 性能

GCM はクライアント/サーバ・アプリケーションなので， Galaxy Configuration Utility (GCU) に比べると， コマンドや構成イベントに対する反応が遅いです。 Galaxy の CPU 割り当ては数ミリ秒しかかからず，GCU にただちに反映されますが， CM では，表示を更新するまでに数秒かかることがあります。 リアルタイム応答が必要なときには，GCU を実行してください (また，複数のハード・パーティションがある場合には， GCU の複数のインスタンスを実行してください)。

Microsoft Windows を実行している PC システムは，典型的なデスクトップ PC に搭載されている物理メモリよりかなり多くの物理メモリを必要とすることがあります。 最高の性能を得る

には，少なくとも 128 MB (おそらくそれ以上) のメモリが必要です。 ディスクの空き容量は，ほとんど関係ありません。

ネイティブの OpenVMS GCM クライアントでは， Windows GCM クライアントほどの性能は得られません。 OpenVMS で GCM クライアントを実行するためには， ユーザ・アカウントにかなりのページファイルを割り当てる必要があります。弊社では，PGFLQUO パラメータを 350000 以上に設定するようにお奨めします。 (これは，OpenVMS 上のすべての Java アプリケーションに適用されます。)

GCM サーバの性能は，一般にネットワークの動作によって左右されます。 GCM サーバは大半の時間をアイドル状態で過ごしますが，他の GCM サーバまたはクライアントにハートビート・トランザクションを発行するために頻繁にウェイクアップします。 GCM サーバは構成変更イベントへの応答として， メモリ内で AlphaServer 構成ツリー構造を再エンコードして，XML でエンコードされた表現をアクティブな GCM サーバおよびクライアントのすべてに送信します。 これによって，一般に約 100 KB のネットワーク・トラフィックが発生します。 複数の GCM サーバを含んでいるアソシエーションでは， それぞれのサーバが GCM クライアントを積極的にサポートしている場合， GCM サーバはこのような突出データを単一の構成モデルにマージして， そのモデルをそれぞれの GCM クライアントに転送しなければなりません。 新しいモデルは 1 MB 以上を必要とすることがあります。

最適な GCM 性能を確保するには，次のことに注意してください。

- 大規模な構成の変更は，通常，GCM サーバがアソシエーションに参加するとき， またはアソシエーションから離脱するときに発生します。 単純なコマンド処理によって生成されるトラフィックは， これよりはるかに少ないです。
- GCM サーバの良好な性能を確保する最良の方法は，アソシエーションの GCM サーバ数を制限することです。
- 複数のアソシエーションを定義することによって， 1 つのアソシエーションのサイズが過度に大きくなるのを防ぐことができます。

14.11 GCM サーバの保守

一般に，GCM サーバの性能メンテナンス作業は必要ありません。 しかし，複数の拡張ロギング機能を有効にした場合， それぞれのログ・ファイルが時間の経過とともに非常に大きくなることがあるので， そのような監査を必要とする場合は， これらのファイルを定期的にチェックする必要があるかもしれません。 このようなログ・ファイルはすべて，SYS$COMMON:[SYS$CONFIG] ディレクトリにあります。 通常の操作モードでは，ロギングは無効になっています。

GCM クライアントについては，ユーザによって検索されたライブラリ・ファイルを時々クリーンアップするだけでよいでしょう。 ライブラリ・ファイルが検索されるたびに， コピーがローカル GCM クライアント・システムに格納されます。

14.12 冗長な GCM サーバ・セットアップの例

このセクションでは，冗長な GCM サーバ・セットアップの例を示します。

```
$ SET DEFAULT SYS$CONFIG
$ DIRECTORY

Directory SYS$COMMON:[SYS$CONFIG]

GCM$SETUP.EXE;1      GCM_BANNER.JPG;1    GCM_CERT.PEM;1       GCM_CUSTOM.XML;1
GCM_NOTICE.HTML;1    GCM_PRIVKEY.PEM;1   GCM_RULESET.XML;1

Total of 7 files.
$ RUN GCM$SETUP

OpenVMS GCM-Server Setup Utility
Copyright 2002, Hewlett Packard Corporation

 This utility initializes the GCM Administrative Database:
```

```
SYS$COMMON:[SYS$CONFIG]GCM_ADMIN.EDB

If you are performing an initial GCM-Server installation that will
create an Association of more than a single server instance, you must
perform the following tasks to assure proper server synchronization:

1) Create the new local database using this utility.
2) Copy the database to all other GCM-Server instances in your Association.
3) Start each GCM-Server.

You can start servers via $@SYS$STARTUP:GCMSRV$STARTUP.COM or by invoking
this utility on each system.  When an existing database is found, this
utility will offer additional server startup options.

Continue (Y/N) {Y}?

Do you prefer full text assistance (Y/N) {N}? y

SERVER DISCOVERY TCP/IP PORT NUMBER:

By default, the GCM-Servers listen for client connections on
TCP/IP Port 4100.  This can be changed, but each server will need
to be restarted, and each client will need to specify the new
port number in their "Server Connection Preferences" settings.

Use the default Port Number 4100 (Y/N) {Y}? y

CONNECTION BOUNDS:

By default, the GCM-Servers support up to 4 concurrent clients
and 8 concurrent servers.  This can be changed, but each server
will need to be restarted. Be advised that GCM performance may
suffer as these values are increased.

Use the default concurrent CLIENT limit of 4 (Y/N) {Y}? y

Use the default concurrent SERVER limit of 8 (Y/N) {Y}? y

CONNECTION SECURITY:

By default, the GCM-Servers expect that their clients will be
using secure connections (SSL).  This can be disabled, but the
servers will need to be restarted, and each client will need
to change their "Server Connection Preferences" settings.

WARNING - DISABLING SECURITY COULD POTENTIALLY COMPROMISE THE
INTEGRITY OF YOUR SYSTEM AND IS NOT RECOMMENDED BY HEWLETT PACKARD.

Use SSL Client-Server Security (Y/N) {Y}? y

%SECURITY - WILL BE ENABLED.

SERVER ASSOCIATION RECORD:

Multiple GCM-Servers can form an "Association" of systems, providing
a wide management scope.  This Association may include one server per
soft-partition on one or more hard-partition on one or more physical
system. Regardless of how many servers are in the Association, you
need to define an Association Name that describes the involved systems.
Choose a simple descriptive string such as "Engineering Lab Systems".

Enter the Association Name (use no quotes): GCM Test Association

SYSTEM RECORDS:

Each system in the Association must be uniquely identified by its
IP Address, System Number, Hard-Partition Number, and Soft Partition
Number.  At least one system must be defined.  You may define multiple
systems now, or define additional systems after establishing your
first client connection.

Enter a fully qualified name for the system running a GCM-Server.
```

```
For example: star.zko.hp.com. When you are done entering system
records, enter 0 at the following prompt.

Enter a SYSTEM Name (ex: star.zko.hp.com) (RETURN when done): wfglx4.zko.hp.com

Enter a fully qualified IP Address for the system running a
GCM-Server.  For example: 16.32.112.16
If you prefer to use DNS to lookup the address, enter 0

Enter a System IP Address (0 to use DNS): 16.32.112.16

The SYSTEM NUMBER is a simple numeric value that uniquely identifies
soft and hard partitions that reside in a common partitionable
computer.  For example, if you have two separate partitionable
computers in your Association, each having its own hard and soft
partitioned instances, enter a SYSTEM NUMBER of 0 for all hard and
soft partitions in the first computer, and enter 1 for those in the
second computer.

Enter a System Number (range 0-7): 0

The HARD PARTITION NUMBER is a numeric value that uniquely identifies
which HARD Partition a GCM-Server resides within.  If a system has
only a single HARD Partition, enter 0.  If a system has multiple
HARD Partitions, use the appropriate Hard Partition ID.  These are
sequential numeric values which were used when creating the system
partitions.  You can also obtain these values by running the Galaxy
Configuration Utility via $ CONFIG GALAXY command.

Enter a HARD PARTITION Number (range 0-7): 0

The SOFT PARTITION NUMBER is a numeric value that uniquely identifies
which SOFT Partition a GCM-Server resides within.  If a system has
only a single SOFT Partition, enter 0.  If a system has multiple
SOFT Partitions, use the appropriate Soft Partition ID.  These are
sequential numeric values which were used when creating the system
partitions.  You can also obtain these values by running the Galaxy
Configuration Utility via $ CONFIG GALAXY command.

Enter a SOFT PARTITION Number (range 0-7): 0

%Define additional system records as needed...

Enter a fully qualified name for the system running a GCM-Server.
For example: star.zko.hp.com. When you are done entering system
records, enter 0 at the following prompt.

Enter a SYSTEM Name (ex: star.zko.hp.com) (RETURN when done): wfglx5.zko.hp.com

Enter a fully qualified IP Address for the system running a
GCM-Server.  For example: 16.32.112.16
If you prefer to use DNS to lookup the address, enter 0

Enter a System IP Address (0 to use DNS): 16.32.112.17

The SYSTEM NUMBER is a simple numeric value that uniquely identifies
soft and hard partitions that reside in a common partitionable
computer.  For example, if you have two separate partitionable
computers in your Association, each having its own hard and soft
partitioned instances, enter a SYSTEM NUMBER of 0 for all hard and
soft partitions in the first computer, and enter 1 for those in the
second computer.

Enter a System Number (range 0-7): 0

The HARD PARTITION NUMBER is a numeric value that uniquely identifies
which HARD Partition a GCM-Server resides within.  If a system has
only a single HARD Partition, enter 0.  If a system has multiple
HARD Partitions, use the appropriate Hard Partition ID.  These are
sequential numeric values which were used when creating the system
partitions.  You can also obtain these values by running the Galaxy
Configuration Utility via $ CONFIG GALAXY command.
```

```
Enter a HARD PARTITION Number (range 0-7): 0

The SOFT PARTITION NUMBER is a numeric value that uniquely identifies
which SOFT Partition a GCM-Server resides within.  If a system has
only a single SOFT Partition, enter 0.  If a system has multiple
SOFT Partitions, use the appropriate Soft Partition ID.  These are
sequential numeric values which were used when creating the system
partitions.  You can also obtain these values by running the Galaxy
Configuration Utility via $ CONFIG GALAXY command.

Enter a SOFT PARTITION Number (range 0-7): 1

%Define additional system records as needed...

Enter a fully qualified name for the system running a GCM-Server.
For example: star.zko.hp.com. When you are done entering system
records, enter 0 at the following prompt.

Enter a SYSTEM Name (ex: star.zko.hp.com) (RETURN when done):

CLIENT AUTHORIZATION RECORDS:

At least one client must be defined in order to allow an initial
client-server connection to be established. Additional clients may
be defined now, or at any point using the client application and the
GCM Subscription Procedure.  This initial client record must provide
fully privileged access for the specified user so that subsequent
GCM administration functions can be performed.

Enter a string which describes this client.
The string can be the users full name, or job title, etc.

Enter Client's Full Name or Title (RETURN when done): First Last

Enter the clients USER name.  This is a single word that identifies
the user, such as their last name.

Enter Client's Username: First

Enter the clients PASSWORD.  This is a unique GCM password,
unrelated to any system authorization function.
Note: Passwords can be changed by any GCM-Client with Admin privilege.

Enter initial Password for First: Last

Enter the clients EMAIL Address.  This is particularly important
for the client that is serving the role of GCM Administrator as
they will receive email subscription requests.

Enter EMAIL Address for First: First.Last@hp.com

CONFIG privilege allows a user to issue commands that alter a system
configuration (if they also have the COMMAND privilege) and to load
and save configuration models.

Give First CONFIG Privilege (Y/N) {Y}? y

COMMAND privilege allows a user to issue DCL commands.

Give First COMMAND Privilege (Y/N) {Y}? y

USER-COMMAND privilege allows a user to create their own command
menu entries. Commands are executed in a privileged context so
use discretion when authorizing this privilege.

Give First USER-COMMAND Privilege (Y/N) {Y}? y

POWER privilege allows a user to issue commands that power on or off
system components for systems that support such operations.

Give First POWER Privilege (Y/N) {Y}? y
```

```
 ADMIN privilege allows a user to modify the GCM-Server Administration
 Database.  Use discretion when authorizing this privilege.

 Give First ADMIN Privilege (Y/N) {Y}? y

 You may choose to ENABLE this client immediately, or enable the
 client later once the GCM is fully configured.
 IMPORTANT: Be sure to enable the initial administrator client!

 Enable First account now (Y/N) {Y}? y

 %Define additional client records as needed...

 Enter a string which describes this client.
 The string can be the users full name, or job title, etc.

 Enter Client's Full Name or Title (RETURN when done):

 SERVER STARTUP OPTIONS:

 You may choose to have the local GCM-Server started automatically upon
 system boot.  If you choose this option, the server will be started
 during the next system boot. To accomplish this, the startup file
 SYS$STARTUP:GCMSRV$STARTUP.COM will be added to the Layered Product
 startup database.

 Do you want the local GCM-SERVER to start on System Boot (Y/N) {Y}? y

 You may choose to start the local GCM-Server now, or you can start
 it later via $@SYS$STARTUP:GCMSRV$STARTUP.COM

 Do you want to start the local GCM-SERVER now (Y/N) {Y}? y


********************** POST SETUP TASKS **************************

 This completes the GCM Admin Database initialization.

 IMPORTANT:

 If you are using multiple GCM-Servers, copy the newly created GCM
 Admin Database file: SYS$COMMON:[SYS$CONFIG]GCM_ADMIN.EDB, to the
 same location on each system you defined in the Association, then
 start or restart each server via $ @SYS$STARTUP:GCMSRV$STARTUP.COM

 If the database has been properly defined, each server will detect
 the presence of the other servers and form the specified Association.
 If the GCM_SERVER process fails to start, examine the server logfile
 SYS$COMMON:[SYS$CONFIG]GCM_SERVER.LOG.

 When the server has started, you may use the GCM-Client to establish
 a connection and further tune the installation.

 For maximum security, you may wish to protect or remove this utility.

$
```

第15章　DCL CLI を使った CPU の再割り当て

OpenVMS では，CPU リソースの管理方法が複数サポートされます。 コンソールはコンソール環境変数を使用して， 各 CPU のデフォルトの所有者インスタンスを設定します。 このため，CPU リソースは静的に割り当てることができ，正確な初期構成が可能です。 コールド・ブート (つまりパワー・サイクルまたは初期化) では， コンソールの不揮発性 RAM からこのデフォルト構成が復元されます。

構成がブートされた後， CMKRNL (Change Mode to Kernel) 特権を持つユーザに対して，OpenVMS ではもっと洗練された手段でリソースを割り当てることができます。 ここではこれらの方法について説明します。

15.1 DCL による再割り当て

CMKRNL 特権を持っている場合，次の DCL コマンドを使用して， CPU の再割当て操作を行うことができます。

```
$ STOP/CPU/MIGRATE=instance-or-id   cpu-id
```

ターゲット・インスタンス名 (SCSNAME) または数値 ID (0，1 など) と， 再割り当てする CPU の数値 ID を指定しなければなりません。 次の例では，このコマンドの形式をいくつか示しています。

```
$ STOP/CPU/MIGRATE=0  4       !Reassign CPU 4 to instance 0
$ STOP/CPU/MIGRATE=1  3,4,5   !Reassign CPUs 3,4,5 to instance 1
$ STOP/CPU 7/MIGRATE=BIGBNG   !ReassignCPU 7 to instance BIGBNG
$ STOP/CPU/ALL/MIGRATE=0      !Reassign all secondary CPUs to instance 0
```

これらのコマンドはコマンド・プロシージャに挿入できます。たとえば，必要とされる処理能力がわかっているアプリケーションのスタートアップ・プロシージャで，余分な CPU リソースをインスタンスに移動することができます。 同様に，時間のかかる I/O 集約型操作 (たとえばバックアップなど) をこれから実行するインスタンスから，CPU の割り当てを解除して，CPU を他のインスタンスが使用できるように設定することもできます。ジョブが完了したら，割り当てを元に戻すことができ，また， シャットダウンされるインスタンスから CPU を再割り当てすることもできます。

この方法では，あるインスタンスからリソースを再割り当てすることだけが可能です。 これは，Galaxy ソフトウェア・アーキテクチャで定義されているプッシュ・モデルです。 このモデルでは，現在の使い方を認識しない他のインスタンスによって，リソースが奪われるのを防止します。 DCL を使用して Galaxy システム全体を効果的に管理するには，関係する各インスタンスにログインするか，SYSMAN ユーティリティを使用して，所有者インスタンスでコマンドを実行する必要があります。

15.2 GCU のドラッグ・アンド・ドロップによる再割り当て

GCU では，Galaxy リソースを管理するために会話型のビジュアル・インタフェースが提供されます。 GCU を使用すると，インスタンス間でドラッグ・アンド・ドロップするだけで，CPU を再割り当てすることができます。さらに，GCU を使用すると， さまざまな構成のチャート (構成モデルと呼びます) を作成し， それをファイルに保存できます。構成モデルはいつでもロードして， 設定することができ，システムは要求されたモデルに設定するために， 必要に応じてリソースを再割り当てします。

15.3 インターモダル再割り当て

Galaxy ソフトウェア・アーキテクチャではリソース・プッシュ・モデルが定義されているため，リソースは， 現在それを所有している Galaxy インスタンスから割り当てなければなりません。 ユーティリティやユーザがマルチインスタンス Galaxy 構成でリソースの割り当てを効

果的に管理するには，各インスタンスでコマンドを実行するための何らかの手段を設定しなければなりません。

このような手段の 1 つとして，各 Galaxy インスタンスでウィンドウまたはターミナル・セッションを開き，これらの各ウィンドウでリソース管理操作を実行する方法があります。

別の方法として，SYSMAN ユーティリティと基礎になる SMI サーバ を使用して， 所有者インスタンスでコマンド環境を設定することもできます。この方法を使用すると，特定のリソース管理操作を実行するためにかなり単純なコマンド・プロシージャを作成できます。しかし，この方法には制限があります。 まず，関係する Galaxy インスタンスはクラスタ内に存在しなければなりません。 また，コマンド・プロシージャは可変パラメータを SYSMAN 環境スクリプトに効果的に渡すことができず，SYSMAN スクリプトの内部にリモート・システム・パスワードを指定することもできません。 したがって， SYSMAN を使用する汎用のコマンド・プロシージャ・インスタンスを作成するのは面倒です。

GCU は実際に，管理アクションを実行するために，可能であれば必ず SYSMAN を使用します。 システムが SYSMAN をサポートするように構成されていない場合は， GCU は管理トランスポートとしてプロキシ・アカウント間で DECnet タスク間通信を使用しようとします。その方法も失敗した場合は (つまり，システムで DECnet が実行されていなかったり， 必要なプロキシ・アカウントが設定されていない場合など)，GCU は， GCU が現在実行されているインスタンス以外の Galaxy インスタンスを管理することができません。 選択した場合は，各 Galaxy インスタンスに対して 1 つずつ， GCU の複数のコピーを実行できます。 しかしほとんどの場合，OpenVMS Galaxy システムはクラスタ接続されているか， DECnet を使用していると考えても構いません。

GCU の管理アクションは，SYS$MANAGER:GCU$ACTIONS.COM コマンド・プロシージャをもとにして実行されます。 このファイルを変更して，それぞれの環境に適するようにアクションをカスタマイズすることができます。 たとえば，TCP/IP 環境では，管理トランスポートのために REXEC や同様のユーティリティを選択することができ，管理アクションが実行されるときに，何らかの形式の通知やログを含むこともできます。

GCU$ACTIONS.COM ファイルは，動作方法が通常のファイルと少し異なります。 SYSMAN を使用する場合，このプロシージャはテンポラリ・ファイルに小さな SYSMAN コマンド・スクリプトを作成して，SYSMAN が取り扱うことができない可変パラメータを取り扱います。SYSMAN を使用できない場合は， このプロシージャは所有者インスタンスのプロキシ・アカウントに対して， DECnet タスク間接続を開始しようとします。接続が成功すると， この接続を使用して，所有者インスタンスにある GCU$ACTIONS.COM のコピーにコマンド・パラメータを渡します。 最終的に，所有者インスタンスがローカルにコマンドを実行します。

15.4 Galaxy サービスを使用したソフトウェアによる再割り当て

おそらくリソース割り当てを管理するための最適な方法は， Galaxy API を使用して独自のリソース管理ルーチンを作成する方法でしょう。独自のルーチンを作成すれば，独自の基準およびアプリケーション環境をもとに， リソース管理に関する判断を下すことができます。 しかし，15.3 項「インターモダル再割り当て」 で説明したものと同じプッシュ・モデルの 制限事項があるため，ルーチンは Galaxy 対応でなければならず， おそらく相互の操作を調整するために共用メモリを使用する必要があります。

$CPU_TRANSITION[W] は，CPU の管理に使用できる OpenVMS システム・サービスです。$CPU_TRANSITION[W] は，現在のシステムの構成セットの中の CPU， または OpenVMS Galaxy 構成の中の未割り当ての CPU の現在の処理状態を変更します。

15.5 再割り当ての失敗

CPU の再割り当てはさまざまな理由で失敗したり，ブロックされることがあります。 GCU は管理アクションを SYSMAN または DCL スクリプトに埋め込むので， 再割り当てが失敗した理由を常に識別して報告できるわけではありません。 たとえば GCU は，プライマリ CPU の再割り当てを禁止するために， 再割り当てアクションを許可する前に，特定のチェックを実行します。また，オペレーティング・システムやコンソール・ファームウェアだけが検出できるその他の理由によって，再割り当てが失敗することもあります。たとえば，現在，プロセス・

アフィニティや Fast Path デューティが割り当てられている CPU の再割り当ての失敗をオペレーティング・システムが検出した場合， DCL メッセージがコンソールとユーザ端末の両方に表示されます。

再割り当てのための Galaxy API は，ほとんどの失敗を呼び出し側に報告することができます。しかし，再割り当てサービスを使用した場合でも，ハードウェア・プラットフォームの依存関係をオペレーティング・システムから確認できないために，コンソールが再割り当てを拒否することがあります。

第16章　Galaxy と NUMA のコマンドとレキシカル関数

この章では，CPU のホット・スワップやホット・アドと Galaxy の設定で使用する DCL コマンドとレキシカル関数を説明します。 また，使い方の例も示します。 詳細は，『OpenVMS DCL ディクショナリ』を参照してください。 コマンドは，次のカテゴリに分類されます。

- CPU 関連の DCL コマンド (表 16-1 「CPU 関連の DCL コマンド」)
- NUMA/RAD 関連の DCL コマンド (表 16-2 「NUMA/RAD 関連の DCL コマンド」)
- Galaxy 関連の DCL コマンド (表 16-3 「Galaxy 関連の DCL コマンド」)
- Galaxy F$GETSYI 項目コード (表 16-4 「Galaxy F$GETSYI 項目コード」)
- パーティショニング F$GETSYI 項目コード (表 16-5 「パーティショニング F$GETSYI 項目コード」)
- SMP F$GETSYI 項目コード (表 16-6 「SMP F$GETSYI 項目コード」)
- RAD F$GETJPI 項目コード (表 16-7 「RAD F$GETJPI 項目コード」)
- RAD F$GETSYI 項目コード (表 16-8 「RAD F$GETSYI 項目コード」)

16.1 CPU 関連の DCL コマンド

これ以降の数ページに記載されている表では，CPU のホット・スワップやホット・アドで使われる DCL コマンドを説明しています。

表 16-1 CPU 関連の DCL コマンド

コマンド	修飾子	説明
SET CPU	/[NO]AUTO_START	指定された CPU のインスタンス固有の自動起動フラグをセットまたはクリアする。
	/[NO]FAILOVER	指定された CPU のインスタンス固有のフェールオーバ関係を定義または削除する。
	/MIGRATE	CPU の所有権を現在のインスタンスから別のソフト・パーティションに移す。
	/POWER	1 つ以上の CPU スロットの電源を投入または切断する。 有効なオプションは，ON と OFF である。
	/REFRESH	ハードウェア構成ツリーを使用して， 指定された CPU の OpenVMS コンテキストを調べ，更新する。
	/OVERRIDE_CHECKS	CPU をアクティブ・セットから削除するのに SET CPU コマンドを使う場合に，指定されたプロセッサを削除できるかどうかを調べる一連のチェックを省略する。
	/START	指定された CPU が OpenVMS アクティブ・セットにまだ参加していない場合は， 参加要求を開始する。
SHOW CPU	/ACTIVE_SET	システムのアクティブ・セットのメンバであるプロセッサだけを表示対象として選択する。
	/CONFIGURE_SET	システムの構成セットのメンバであるプロセッサ，すなわち，現在のインスタンスによってアクティブに所有され， 制御されているプロセッサだけを表示対象として選択する。
	/POTENTIAL_SET	システムの潜在的なセットのメンバであるプロセッサ，すなわち， ハード・パーティション内の CPU のうち，現在のインスタンスのアクティブ・セットに参加するための条件を満たしている CPU だけを表示対象として選択する。 このセットに含まれていても，その CPU が現在のインスタンスによって所有されている (または，いずれ所有される) ことを意味するわけではない。潜在的なセットは，物理的に存在する CPU のうち，使用可能になった場合に， インスタンス固有のハードウェアおよびソフトウェア互換性制約を現時点で 満たしているものを記述するにすぎない。

表 16-1 CPU 関連の DCL コマンド (続き)

コマンド	修飾子	説明
	/STANDBY_SET	システムのスタンバイ・セットのメンバであるプロセッサ，すなわち，ハード・パーティション内の CPU のうち，ソフト・パーティションによって所有されていず，現在のインスタンスによる所有が可能なものだけを表示対象として選択する。
	/SYSTEM	現在のインスタンスに関するプラットフォーム固有のハードウェア情報を表示する。
START/CPU	/POWER[=ON]	CPU をアクティブ・セットに入れる前に，CPU に電源を投入する。
STOP/CPU	/MIGRATE	CPU の所有権を現在のインスタンスから別のソフト・パーティションに移す。
	/POWER=OFF	CPU をアクティブ・セットから削除した後，CPU の電源を切断する。

表 16-2 NUMA/RAD 関連の DCL コマンド

コマンド	説明
INITIALIZE/RAD=n	キューに割り当てられたバッチ・ジョブを実行する RAD 番号を指定する。
SET ENTRY/RAD=n	発行されたバッチ・ジョブを実行する RAD 番号を指定する。
SET PROCESS/RAD=HOME=n	プロセスのホーム RAD を変更する。
SET QUEUE/RAD=n	キューに割り当てられたバッチ・ジョブを実行する RAD 番号を指定する。
SHOW PROCESS/RAD	ホーム RAD を表示する。
START QUEUE/RAD=n	キューに割り当てられたバッチ・ジョブを実行する RAD 番号を指定する。
SUBMIT/RAD=n	発行されたバッチ・ジョブを実行する RAD 番号を指定する。

表 16-3 Galaxy 関連の DCL コマンド

コマンド	説明
CONFIGURE GALAXY	OpenVMS Galazy システムの監視，表示，操作を行うために，Galaxy 構成ユーティリティ (GCU) を起動する。
INSTALL/LIST	標準のグローバル・セクションだけでなく，Galaxywide セクションも返す。
SHOW MEMORY/PHYSICAL	システムのメモリ使用状況を表示する。

16.2 Galaxy 関連の DCL レキシカル関数

ここでは，Galaxy，パーティショニング，SMP，および RAD の設定で使われる DCL レキシカル関数 F$GETSYI および F$GETJPI の項目コードを示します。

表 16-4 Galaxy F$GETSYI 項目コード

項目コード	タイプ	説明
GLX_MAX_MEMBERS	整数値	現在の Galaxy 構成に参加できる最大インスタンス数を返す。
GLX_FORMATION	文字列	このインスタンスが属する Galaxy 構成の作成日時を示す タイムスタンプ文字列を返す。

表 16-4 Galaxy F$GETSYI 項目コード (続き)

項目コード	タイプ	説明
GLX_TERMINATION	文字列	このインスタンスが最後に属していた Galaxy 構成の終了日時を示す タイムスタンプ文字列を返す。
GLX_MBR_NAME	文字列	Galaxy メンバシップの既知の名前を示す文字列を返す。
GLX_MBR_MEMBER	整数値	64 バイトの整数を返す。それぞれの 8 バイトは Galaxy メンバを表し，7 から 0 までリストされる。インスタンスがメンバである場合は値は 1 であり， メンバでない場合は 0 である。

表 16-5 パーティショニング F$GETSYI 項目コード

項目コード	タイプ	説明
HP_ACTIVE_CPU_CNT	整数値	ファームウェア・コンソール・モードではないハード・パーティション内の CPU 数を返す。 OpenVMS では，これは CPU がハード・パーティション内のインスタンスの 1 つのアクティブ・セットに参加しているか参加プロセス中であることを意味する。
HP_ACTIVE_SP_CNT	整数値	ハード・パーティション内で実行中のアクティブなオペレーティング・システム・インスタンス数を返す。
HP_CONFIG_SP_CNT	整数値	現在のハード・パーティション内のソフト・パーティションの最大数を返す。この数は，オペレーティング・システム・インスタンスが特定のソフト・パーティション内で実行中であることを意味しない。
HP_CONFIG_SBB_CNT	整数値	現在のハード・パーティション内の既存のシステム・ビルディング・ブロック数を返す。

表 16-6 SMP F$GETSYI 項目コード

項目コード	タイプ	説明
ACTIVE_CPU_MASK	整数値	CPU インデックス付きのビットベクトルを表す値を返す。特定のビット位置がセットされているとき，その CPU ID 値を持つプロセッサは，インスタンスのアクティブ・セットのメンバである。すなわち，OpenVMS SMP スケジューリング活動に参加している。
AVAIL_CPU_MASK	整数値	CPU インデックス付きのビットベクトルを表す値を返す。特定のビット位置がセットされているとき，その CPU ID 値を持つプロセッサは，インスタンスの構成セットのメンバである。すなわち，パーティションによって所有され，発行元インスタンスによって制御されている。
POTENTIAL_CPU_MASK	整数値	CPU インデックス付きのビットベクトルを表す値を返す。特定のビット位置がセットされているとき，その CPU ID 値を持つプロセッサは， インスタンスの潜在的なセットのメンバである。潜在的なセット内の CPU は， このインスタンスによって所有された場合， このインスタンスの VMS アクティブ・セットに積極的に参加できることを意味する。このルールに従うには，CPU の特性がそのインスタンスについて特に定義されているハードウェアおよびソフトウェア互換性ルールに一致しなければならない。
POWERED_CPU_MASK	整数値	CPU インデックス付きのビットベクトルを表す値を返す。特定のビット位置がセットされているとき，その CPU ID 値を持つプロセッサは，インスタンスのパワード・セットのメンバである。すなわち，ハード・パーティション内に物理的に存在し，操作のために起動された CPU である。

表 16-6 SMP F$GETSYI 項目コード (続き)

項目コード	タイプ	説明
PRESENT_CPU_MASK	整数値	CPU インデックス付きのビットベクトルを表す値を返す。特定のビット位置がセットされているとき，その CPU ID 値を持つプロセッサは，インスタンスのプレゼント・セットのメンバである。すなわち，ハード・パーティション内に物理的に存在する CPU である。
CPUCAP_MASK	文字列	コンマで区切られ，CPU ID のインデックスが付けられた 16 進値のリストを返す。それぞれの値は，ビットベクトルを表す。セットされているとき，対応するユーザ機能がその CPU で使用可能である。
PRIMARY_CPUID	整数値	この OpenVMS インスタンスのプライマリ・プロセッサの CPU ID を返す。
MAX_CPUS	整数値	このインスタンスが認識できる最大 CPU 数を返す。
CPU_AUTOSTART	整数値	コンマで区切られ，CPU ID のインデックスが付けられた 0 と 1 のリストを返す。 値 1 のエントリは，特定の CPU が外部から現在のインスタンスに移行した場合，またはすでに所有されているときに起動された場合， VMS アクティブ・セットに参加することを示す。
CPU_FAILOVER	整数値	コンマで区切られ， CPU ID のインデックスが付けられたパーティション ID 番号のリストを返す。 このリストは，現在のインスタンスがクラッシュした場合のプロセッサのデスティネーションを定義している。
POTENTIALCPU_CNT	整数値	このインスタンスの潜在的なセットのメンバであるハード・パーティション内の CPU 数。 潜在的なセット内の CPU は，このインスタンスによって所有された場合，このインスタンスの VMS アクティブ・セットに積極的に参加することを意味する。このルールに従うには，CPU の特性がそのインスタンスについて特に定義されたハードウェアおよびソフトウェア互換性ルールに一致しなければならない。
PRESENTCPU_CNT	整数値	ハードウェア・スロットに物理的に常駐するハード・パーティション内の CPU 数。
POWEREDCPU_CNT	整数値	物理的に起動されているハード・パーティション内の CPU 数。

表 16-7 RAD F$GETJPI 項目コード

項目コード	タイプ	説明
HOME_RAD	整数値	ホーム・リソース・アフィニティ・ドメイン (RAD)。

表 16-8 RAD F$GETSYI 項目コード

項目コード	タイプ	説明
RAD_CPUS	整数値	コンマによって区切られた RAD, CPU のペアのリストを返す。
RAD_MEMSIZE	整数値	コンマによって区切られた RAD, PAGES のペアのリストを返す。
RAD_MAX_RADS	整数値	このプラットフォーム上で可能な最大 RAD 数を返す。値は常に 8 である (物理的に存在するクォド・ビルディング・ブロック (QBB) 数に関係なく)。
RAD_SHMEMSIZE	整数値	コンマで区切られた RAD, PAGES のペアのリストを返す。

16.3 DCL コマンド例

ここで説明する DCL コマンドとレキシカル関数は，OpenVMS Galaxy を管理するのに役立ちます。 次の順序で，例を挙げて説明します。

- STOP/CPU/MIGRATE (16.3.1.1 項 「STOP/CPU/MIGRATE」を参照)
- SHOW CPU (16.3.1.2 項 「SHOW CPU」を参照)
- SET CPU (16.3.1.3 項 「SET CPU」を参照)
- SHOW MEMORY (16.3.2 項 「SHOW MEMORY」を参照)
- レキシカル関数 (16.3.3 項 「レキシカル関数の例」を参照)
- INSTALL LIST (16.3.4 項 「INSTALL LIST」を参照)
- INSTALL ADD (16.3.5 項 「INSTALL ADD/WRITABLE」を参照)
- CONFIGURE GALAXY (16.3.6 項 「CONFIGURE GALAXY」を参照)

これらのコマンドについては，次の節で説明します。

16.3.1 CPU コマンド

CPU は OpenVMS Galaxy で割り当て可能なリソースです。 CPU を割り当てるには， Physical Structure チャート (13.4.2 項「Physical Structure チャート」 を参照) または STOP/CPU/MIGRATE コマンド (16.3.1.1 項 「STOP/CPU/MIGRATE」 を参照) を使用します。

16.3.1.1 STOP/CPU/MIGRATE

STOP/CPU/MIGRATE コマンドは， CPU の所有権を現在のインスタンスから別のソフト・パーティションに移します。

たとえば，次のコマンドを入力します。

```
$ STOP/CPU/MIGRATE=GLXSYS  4
```

この例では，GLXSYS はインスタンス名またはパーティション ID 値です。 ターゲット上でオペレーティング・システムが実行している必要はありません。

次のメッセージが端末に表示されます。

```
%SYSTEM-I-CPUSTOPPING, trying to stop CPU 4 after it reaches quiescent state
```

ソース・コンソールには，次のメッセージが表示されます。

```
%SMP-I-STOPPED, CPU #04 has been stopped.
```

デスティネーション・コンソールには，次のメッセージが表示されます。

```
%SMP-I-SECMSG, CPU #04 message:   P04>>>START
%SMP-I-CPUTRN, CPU #04 has joined the active set.
```

16.3.1.2 SHOW CPU

SHOW CPU コマンドは，指定されたプロセッサの状態，属性， 機能に関する情報を表示します。たとえば，次のように表示されます。

```
$ show cpu

System: GLXSYS, Compaq AlphaServer GS320 6/731

CPU ownership sets:
   Active               0-31
   Configure            0-31

CPU state sets:
   Potential            0-31
```

```
   Autostart            0-31
   Powered Down         None
   Failover             None

$ show cpu/system

System: GLXSYS, Compaq AlphaServer GS320 6/731

  SMP execlet   = 2 : Enabled : Full checking.
  Config tree   = Version 6
  Primary CPU   = 0
  HWRPB CPUs    = 32
  Page Size     = 8192
  Revision Code =
  Serial Number = BUDATEST
  Default CPU Capabilities:
        System: QUORUM RUN
  Default Process Capabilities:
        System: QUORUM RUN

CPU ownership sets:
   Active               0-31
   Configure            0-31

CPU state sets:
   Potential            0-31
   Autostart            0-31
   Powered Down         None
   Failover             None
```

16.3.1.3 SET CPU

SET CPU コマンドは，指定された CPU に関するユーザ機能を変更します。

次のコマンドの *n* は，CPU 番号，コンマで区切られた CPU のリスト， または /ALL 修飾子を表します。

- SET CPU *n*/FAILOVER=**y**

  インスタンスのポテンシャル・セット内の各 CPU に対して， インスタンス固有のフェールオーバ環境を設定します。 インスタンスがクラッシュすると， 現在のインスタンス以外のフェールオーバ・ターゲットを持つ CPU は， そのターゲットに割り当てられるか，移行されます。

  y は，インスタンスの名前，または現在のハード・パーティション内の物理パーティション ID です。

- SET CPU n/NOFAILOVER

  指定された CPU のインスタンス固有のフェールオーバ関係を削除します。

- SET CPU **n**/AUTO_START

  指定された CPU のインスタンス固有のオートスタート・フラグをセットまたはクリアします。

  オートスタートが有効な場合，パーティションに割り当てられるか移行されたときに，その CPU は OpenVMS アクティブ・セットに参加します。 また，その CPU が発行元インスタンスによって所有されているときに起動遷移が完了した場合も自動起動します。

- SET CPU n/NOAUTO_START

  指定された CPU のインスタンス固有のオートスタート・フラグをクリアします。

16.3.2 SHOW MEMORY

SHOW MEMORY コマンドは，システムのメモリの使用状況を表示します。 たとえば，次のように表示されます。

```
$ SHOW MEMORY/PHYSICAL

            System Memory Resources on 5-OCT-2001 20:50:19.03

Physical Memory Usage (pages):     Total        Free      In Use    Modified
   Main Memory (2048.00Mb)        262144        228183    31494      2467
Of the physical pages in use, 11556 pages are permanently allocated to OpenVMS.


$ SHOW MEMORY/PHYSICAL

              System Memory Resources on 5-OCT-2001 07:55:14.68

Physical Memory Usage (pages):     Total        Free      In Use    Modified
   Private Memory (512.00Mb)       65536        56146      8875         515
   Shared Memory (1024.00Mb)       131072       130344      728

Of the physical pages in use, 6421 pages are permanently allocated to OpenVMS.
$
```

16.3.3 レキシカル関数の例

レキシカル関数は，現在のプロセスの文字列と属性に関する情報を返す関数です。次のコマンド・プロシージャは，F$GETSYI レキシカル関数を使用して， Galaxy システムの属性を表示するコマンド・プロシージャを作成します。

レキシカル関数のコマンド・プロシージャの例

```
$ create shoglx.com
$ write sys$output ""
$ write sys$output "Instance = ",f$getsyi("scsnode")
$ write sys$output "Platform = ",f$getsyi("galaxy_platform")
$ write sys$output "Sharing Member = ",f$getsyi("galaxy_member")
$ write sys$output "Galaxy ID = ",f$getsyi("galaxy_id")
$ write sys$output "Community ID = ",f$getsyi("community_id")
$ write sys$output "Partition ID = ",f$getsyi("partition_id")
$ write sys$output ""
$ exit
$ ^Z
```

レキシカル関数のコマンド・プロシージャの出力

```
$ @SHOGLX

Instance = COBRA2
Platform = 1
Sharing Member = 1
Galaxy ID = 5F5F30584C47018011D3CC8580F40383
Community ID = 0
Partition ID = 0

$
```

16.3.4 INSTALL LIST

INSTALL LIST コマンドは，標準のグローバル・セクションだけでなく， galaxywide セクションも返すようになりました。

16.3.5 INSTALL ADD/WRITABLE

INSTALL ADD/WRITABLE コマンドは，標準グローバルだけでなく， Galaxy セクションもサポートするようになりました。 INSTALL ADD/WRITABLE=GALAXY は， 指定されたファイルを書き込み可能な既知のイメージとして Galaxy グローバル・セクションにインストールします。

16.3.6 CONFIGURE GALAXY

CONFIGURE GALAXY コマンドは， Galaxy Configuration ユーティリティ (GCU) を起動して，OpenVMS Galaxy システムを監視し，表示し，会話します。 GCU を使用するには，DECwindows Motif V1.2-4 またはそれ以上と OpenVMS Alpha V7.2 またはそれ以上が必要です。

オプションの model パラメータは， ロードおよび表示する Galaxy 構成モデルの場所と名前を指定します。 model が指定されていないときに， システムが OpenVMS Galaxy として動作している場合は， 現在のアクティブ構成が表示されます。

システムが OpenVMS Galaxy として動作していない場合は， GCU はシングル・インスタンス OpenVMS Galaxy システムを作成するユーザを支援します。

OpenVMS Galaxy 構成モデルは Galaxy Configuration ユーティリティを使用して作成されます。 詳細については，GCU のオンライン・ヘルプを参照してください。 詳細については，GCU または GCM のオンライン・ヘルプを参照してください。

形式：

CONFIGURE GALAXY [model.GCM]

パラメータ:

`GALAXY [model.GCM]`

ロードおよび表示する Galaxy 構成モデルの場所と名前を指定します。

model を指定しなかったときに， システムが OpenVMS Galaxy として動作している場合は，現在のアクティブ構成が表示されます。

修飾子:

`/ENGAGE`

GCU は，グラフィカル・ユーザ・インタフェースを表示せずに，指定された OpenVMS Galaxy 構成モデルに設定 (ロード/確認，およびアクティブ化) します。 確認の後，指定されたモデルがアクティブ・システム構成になります。この修飾子を使用すると，システムをブートした後で， どのような動的リソースの再割り当てが実行されていたとしても， システム管理者はそれとは無関係に， OpenVMS Galaxy システムを既知の構成に復元することができます。 このコマンドを DCL コマンド・プロシージャに埋め込んでおけば， 構成操作を自動化することができます。

`/VIEW`

この修飾子を /ENGAGE および model パラメータと組み合わせて使用すると， GCU は指定された構成モデルをロードし，確認し，アクティブ化し，表示します。

例:

```
$ CONFIGURE GALAXY
```

GCU のグラフィカル・ユーザ・インタフェースを表示します。 システムが現在，OpenVMS Galaxy として構成されている場合は， アクティブ・システム構成が表示されます。

```
$ CONFIGURE GALAXY model.GCM
```

GCU のグラフィカル・ユーザ・インタフェースを表示します。指定された OpenVMS Galaxy 構成モデルがロードされ，表示されますが，ユーザが設定することを選択するまで，アクティブ構成にはなりません。

```
$ CONFIGURE GALAXY/ENGAGE model.GCM
```

GCU コマンド行インスタンスを起動して，GCU のグラフィカル・ユーザ・インタフェースを表示せずに， 指定された OpenVMS Galaxy 構成モデルに設定します。

```
$ CONFIGURE GALAXY/ENGAGE/VIEW model.GCM
```

GCU コマンド行インタフェースを起動して， 指定された OpenVMS Galaxy 構成モデルに設定し， GCU のグラフィカル・ユーザ・インタフェースを表示します。

第17章　共用メモリを使った通信

この章では，次の 2 つの OpenVMS 内部機能について説明します。 これらの機能は，共用メモリを使用して， OpenVMS Galaxy コンピューティング環境のインスタンス間で通信を行います。

- 共用メモリ・クラスタ・インターコネクト (SMCI)
- ローカル・エリア・ネットワーク (LAN) 共用メモリ・デバイス・ドライバ

17.1 共用メモリ・クラスタ・インターコネクト (SMCI)

共用メモリ・クラスタ・インターコネクト (SMCI) は， Galaxy インスタンス間で通信するためのシステム通信サービス (SCS) ポートです。 OpenVMS インスタンスが Galaxy および OpenVMS Cluster メンバの両方としてブートされると，SMCI ドライバがロードされます。この SCS ポート・ドライバは， 共用メモリを介して同じ Galaxy 内の他のクラスタ・インスタンスと通信します。 この機能は，OpenVMS Galaxy ソフトウェア・アーキテクチャが備えている性能向上のための主要な機能の 1 つです。 共用メモリを介してクラスタ接続された他のインスタンスと通信できる機能は，従来のクラスタ・インターコネクトより大幅に性能を向上することができます。

以下のセクションでは， SMCI とともに使用されるドライバとデバイスについて説明します。

17.1.1 SYS$PBDRIVER ポート・デバイス

Galaxy およびクラスタ・メンバの両方としてブートすると， デフォルトで SYS$PBDRIVER がロードされます。 このドライバをロードすると，デバイス PBA*x* が作成されます。 ただし，*x* は Galaxy パーティション ID です。 他のインスタンスがブートされると，PBA*x* デバイスも作成されます。SMCI は他のインスタンスをただちに識別し，そのインスタンスへの通信チャネルを作成します。 従来のクラスタ・インターコネクトと異なり， 他のインスタンスと通信するために新しいデバイスが作成されます。このデバイスにも PBA*x* という名前が割り当てられます。ただし，*x* は，このデバイスの通信相手であるインスタンスの Galaxy パーティション ID です。

たとえば，MILKY と WAY という 2 つのインスタンスで構成される OpenVMS Galaxy について考えてみましょう。 MILKY はインスタンス 0 であり，WAY はインスタンス 1 です。 ノード MILKY がブートされると，デバイス PBA0 が作成されます。 ノード WAY がブートされると， PBA1 が作成されます。2 つのノードが相互に相手を検出すると， MILKY は WAY と通信するために PBA1 を作成し， WAY は MILKY と通信するために PBA0 を作成します。

```
MILKY                   WAY

PBA0:                   PBA1:

PBA1:   <-------->      PBA0:
```

17.1.2 1 つの Galaxy 内の複数のクラスタ

SYS$PBDRIVER は同じ Galaxy 内で複数のクラスタをサポートできます。 この機能は，SYS$PEDRIVER を使用して， 同じ LAN で複数のクラスタがサポートされるのと同じ方法で実現されます。 SYS$PEDRIVER で使用されるクラスタ・グループ番号とパスワードは， 同じ Galaxy コミュニティ内の異なるクラスタを区別するために， SYS$PBDRIVER でも使用されます。 Galaxy インスタンスが LAN を介して他の OpenVMS インスタンスとクラスタ接続もされている場合は， クラスタ・グループ番号は CLUSTER_CONFIG によって適切に設定されます。 現在のクラスタ・グループ番号を判断するには，次のコマンドを入力します。

```
$ MCR SYMAN
SYSMAN> CONFIGURATION SHOW CLUSTER_AUTHORIZATION
Node: MILKY   Cluster group number: 0
```

```
Multicast address: xx-xx-xx-xx-xx-xx
SYSMAN>
```

LAN を介してクラスタ接続されていないときに，同じ Galaxy コミュニティ内で複数のクラスタを稼働させる場合は， クラスタ・グループ番号を設定しなければなりません。 グループ番号とパスワードは，次のように同じクラスタ内のすべての Galaxy インスタンスで同一でなければなりません。

```
$ MCR SYSMAN
SYSMAN> CONFIGURATION SET
CLUSTER_AUTHORIZATION/GROUP_NUMBER=222/PASSWORD=xxxx
SYSMAN>
```

Galaxy インスタンスが LAN を介してクラスタ接続もされている場合は， CLUSTER_CONFIG はクラスタ・グループ番号を質問し， Galaxy インスタンスはこれらのグループ番号を使用します。 LAN を介してクラスタ接続されていない場合は， グループ番号のデフォルトは 0 になります。 つまり，Galaxy 内のすべてのインスタンスは同じクラスタ内に存在します。

17.1.3 SYS$PBDRIVER の SYSGEN パラメータ

ほとんどの場合，SYS$PBDRIVER のデフォルト設定を使用すると適切です。 しかし，いくつかの SYSGEN パラメータも準備されています。 SMCI_PORTS と SMCI_FLAGS の 2 つの SYSGEN パラメータは SYS$PBDRIVER を制御します。

17.1.3.1 SMCI_PORTS

SYSGEN パラメータ SMCI_PORTS は，SYS$PBDRIVER の初期ロードを制御します。 このパラメータはビットマスクであり， ビット 0～25 はそれぞれコントローラ名を表します。 ビット 0 がセットされている場合は，PBAx がロードされます。 これはデフォルト設定です。 ビット 1 がセットされている場合は，PBBx がロードされ， 以下同様で，ビット 25 がセットされている場合は， PBZx がロードされます。 OpenVMS Alpha バージョン 7.3-1 以降の場合，このパラメータはデフォルト値の 1 のままにしておいてください。

追加ポートをロードすると，Galaxy インスタンス間で複数のパスを使用できます。OpenVMS Alpha バージョン 7.3-1 では，SYS$PBDRIVER は， Fast Path をサポートしないので， 複数の通信チャネルを使用しても利点がありません。OpenVMS の将来のリリースでは，SYS$PBDRIVER 用に Fast Path がサポートされるようになる予定です。 Fast Path がサポートされるようになれば， 複数の CPU が割り当てられたインスタンスは， インスタンス間に複数の通信チャネルを持つことで，スループットを向上できます。

17.1.4 SMCI_FLAGS

SYSGEN パラメータ SMCI_FLAGS は，SYS$PBDRIVER の動作を制御します。 現在定義されているフラグはビット 1 だけです。ビット 1 は，ポート・デバイスがそれ自体との通信をサポートするかどうかを制御します。 それ自体との SCS 通信をサポートする機能は，主にテストのために使用されます。デフォルト設定では，このビットはオフに設定され，システム・リソースを節約するために， ローカルな SCS 通信のサポートは無効になります。このパラメータは動的であり， このビットをオンにすると，SCS 仮想サーキットがただちに作成されます。

次の表に，SMCI_FLAGS パラメータのビットとビット・マスクの値を示します。

ビット	マスク	説明
0	0	0 = ローカル通信チャネルを作成しない (SYSGEN のデフォルト)。 ローカル SCS 通信は主にテストで使用され，通常の操作では必要ない。 このビットをオフにしておけば，リソースとオーバヘッドを削減できる。 1 = ローカル通信チャネルを作成する。
1	2	0 = Galaxy およびクラスタの両方としてブートされた場合は， SYS$PBDRIVER をロードする (SYSGEN のデフォルト)。 1 = Galaxy としてブートされた場合は，SYS$PBDRIVER をロードする。
2	4	0 = 必要最低限のコンソール出力 (SYSGEN のデフォルト)。 1 = 完全なコンソール出力，SYS$PBDRIVER は，通信チャネルを作成するときと， 通信チャネルを破棄するときに，コンソール・メッセージを表示する。

17.2 LAN 共用メモリ・デバイス・ドライバ

OpenVMS Galaxy インスタンス間のローカル・エリア・ネットワーク (LAN) 通信は， イーサネット LAN 共用メモリ・ドライバでサポートされます。 この LAN ドライバは共用メモリを介して， 同じ OpenVMS Galaxy システム内の他のインスタンスと通信します。 共用メモリを介して他のインスタンスと通信すれば， 従来の LAN を介した通信より性能を大幅に向上できます。

LAN 共用メモリ・ドライバ SYS$EBDRIVER をロードするには， 次のコマンドを入力します。

```
$ MCR SYSMAN
SYSMAN> IO CONN EBA/DRIVER=SYS$EBDRIVER/NOADAPTER
```

OpenVMS バージョン 7.3-1 以降で，LAN プロトコルがこの LAN デバイス (EBA**n**，ただし **n** はユニット番号) を介して自動的に起動するように設定するには，このドライバをロードするためのプロシージャを構成プロシージャに追加する必要があります。

```
SYS$MANAGER:SYCONFIG.COM.
```

LAN ドライバは Ethernet/IEEE 802.3 と同じフレーム形式でイーサネット LAN をエミュレートしますが， 最大フレーム・サイズは 1518 から 7360 バイトに拡大しています。 LAN ドライバは標準の OpenVMS QIO および VCI インタフェースをアプリケーションに提供します。 既存の QIO および VCI LAN アプリケーションはすべて， 変更せずに動作するはずです。

将来のリリースでは， SYS$EBDRIVER デバイス・ドライバは自動的にロードされるようになります。

第18章　共用メモリ・プログラミング・インタフェース

共用メモリ・グローバル・セクション は，共用コミュニティ内のすべてのインスタンスからアクセスできるメモリをマッピングします。これらのオブジェクトは **Galaxy ワイドの共用セクション** とも呼びます。このような各オブジェクトには名前，バージョン，保護属性があります。

18.1 共用メモリの使用

アプリケーション・プログラムは，Galaxy-wide 共用セクションをマッピングすることで共用メモリにアクセスします。プログラミング・モデルは標準の OpenVMS グローバル・セクションと同じです。つまり，共用メモリを使用する各インスタンスで，その共用メモリを作成，マッピング，アンマッピング，削除します。共用メモリ・グローバル・セクションには次の属性があります。

- ページはデマンド 0 から始まり，あらかじめ共用 PFN に割り当てられます。
- ページはワーキング・セットの一部としてカウントされません。
- ページがプロセスのページ・テーブルで有効になった後，削除されるまで有効なままです。共用メモリ・セクション・ページがディスクにページングされることはありません。
- 共用メモリにアクセスする各インスタンスで共用セクションを作成しなければなりません。
- セクションは一時的でも永久的でも構いません。
- セクションはグループ・セクションまたはシステム・グローバル・セクションになることができます。
- Galaxy-wide 共用セクションは通常のグローバル・セクションと異なるネーム・スペースを使用します。
- ident_64 フィールドに指定されるセクション・バージョンは，Galaxy 全体で有効であるかどうかチェックされます。
- 指定された名前および UIC グループが割り当てられた共用セクションは，共用コミュニティ内に 1 つだけしか存在できません。これは，複数のバージョンが共存できる伝統的なグローバル・セクションとは異なります。
- 共用メモリ・セクションを作成するには，SHMEM 特権が必要です。

プログラマの立場から見ると，共用メモリ・グローバル・セクションはメモリ常駐セクションによく似ています。メモリ常駐セクションを作成するときに使用するシステム・サービスと同じサービスを使用して，Galaxy-wide 共用セクションを作成します。フラグ SEC$M_SHMGS をセットすると，サービスは共用メモリ・グローバル・セクションに対して動作します。

メモリ常駐セクションと異なり，Galaxy-wide セクションに対して空間を割り当てるために，予約メモリ・レジストリは使用されません。SYSMAN RESERVE コマンドは，ノード固有メモリにのみ影響します。通常の OpenVMS ページング操作のために共用メモリは使用されないので，共用メモリを予約する必要はありません。

Galaxy-wide セクションに対して共用ページ・テーブルを作成しなければならないかどうかを指定するためのユーザ・インタフェースもありません。その代わり，Galaxy-wide セクションに対する共用ページ・テーブルの作成は，セクション・サイズによって決定されます。OpenVMS バージョン 7.2 では，共用ページ・テーブルは 128 ページ (1MB) 以上のセクションに対して作成されます。Galaxy-wide 共用ページ・テーブルはすべての Galaxy インスタンス間で共用されます。

18.2 システム・サービス

ここでは，共用メモリ・グローバル・セクションをサポートする新しいシステム・サービスと，変更されたシステム・サービスについて説明します。

18.2.1 強化されたサービス

新しい共用メモリ・グローバル・セクション・フラグ SEC$M_SHMGS を認識するように，次のシステム・サービスが強化されました。

- SYS$CRMPSC_GDZRO_64
- SYS$CREATE_GDZRO
- SYS$MGBLSC_64
- SYS$DGBLSC

次のシステム・サービスは共用メモリを操作できるように強化されていますが，インタフェースは変更されていません。

- SYS$DELTVA
- SYS$DELTVA_64
- SYS$CREATE_BUFOBJ
- SYS$CREATE_BUFOBJ_64
- SYS$DELETE_BUFOBJ

18.2.2 新しいセクション・フラグ SEC$M_READ_ONLY_SHPT

新しいセクション・フラグ SEC$M_READ_ONLY_SHPT は， SYS$CREATE_GDZRO と SYS$CRMPSC_GDZRO_64 サービスで認識されます。このビットがセットされている場合は，読み込みアクセスだけが可能な共用ページ・テーブルをセクション用に作成することをシステムに要求します。この機能は特に，メモリ常駐セクションや Galaxy 共用セクションが多くのプログラムで読み込まれるものの， 書き込むプログラムは 1 つしかない環境で役立ちます。

共用ページ・テーブル・セクションが割り当てられている Galaxy 共用セクションまたはメモリ常駐セクションをマッピングする場合には，データにアクセスするために次のオプションを使用できます。

共用ページ・テーブル	読み込み専用	読み込みと書き込み
何も作成されない	マップ要求に SEC$M_WRT フラグをセットしない。	マップ要求に SEC$M_WRT フラグをセットする。
	プライベート・ページ・テーブルは，セクションをマッピングする共用ページ・テーブル領域を指定した場合でも，常に使用される。	プライベート・ページ・テーブルは，セクションをマッピングする共用ページ・テーブル領域を指定した場合でも，常に使用される。
書き込みアクセス	マップ要求に SEC$M_WRT フラグをセットしない。	マップ要求に SEC$M_WRT フラグをセットする。
	プライベート・ページ・テーブルが使用されることを確認する。 セクションをマッピングする共用ページ・テーブル領域を指定しない。 指定すると，エラー状態 S$_IVSECFLG が返される。	セクションをマッピングする共用ページ・テーブル領域を指定した場合， マッピングのために共用ページ・テーブル・セクションが使用される。
読み込みアクセス	マップ要求に SEC$M_WRT フラグをセットしない。セクションをマッピングする共用ページ・テーブル領域を指定した場合， マッピングのために共用ページ・テーブル・セクションが使用される。	マップ要求に SEC$M_WRT フラグをセットする。 プライベート・ページ・テーブルが使用されることを確認する。セクションをマッピングする共用ページ・テーブル領域を指定しない。 指定すると，エラー状態 SS$_IVSECFLG が返される。

注意:

Galaxy 共用セクションの共用ページ・テーブルも Galaxy 共用セクションとして実装されます。 このため，このセクションに接続されているすべての OpenVMS インスタンスで読み込みアクセスが許可されるか，またはすべてのインスタンスで読み込みアクセスと書き込みアクセスが許可されます。 セクションを作成するために， 最初のインスタンスが要求する SEC$M_READ_ONLY_SHPT フラグの設定がすべてのインスタンスで使用されます。

SYS$CRMPSC_GDZRO_64 サービスを使用すると，常に SEC$M_WRT フラグがセットされ，書き込みのためにセクションをマッピングすることが示されます。 このサービスを使用して，読み込み専用アクセスのために共用ページ・テーブルを使用するセクションを作成する場合は， プライベート・ページ・テーブルを使用しなければならず， セクションをマッピングする共用テーブル領域を指定することはできません。

18.3 Galaxy-wide グローバル・セクション

Galaxy 共用メモリにオブジェクトを作成するには，SHMEM 特権が必要です。 既存のセクションにマッピングする権利は，通常のアクセス制御機能によって制御されます。 既存のセクションをマッピングするために SHMEM は必要ありません。 通常のメモリ常駐セクションを作成する場合は， VMS$MEM_RESIDENT_USER 識別子が必要ですが， Galaxy-wide セクションの場合は必要ありません。

Galaxy-wide メモリ・セクションの作成とマッピングは， メモリ常駐セクションを作成する場合と同じサービスで実行されます。 現在，次のサービスが SEC$M_SHMGS フラグを認識します。

- SYS$CREATE_GDZRO
- SYS$CRMPSC_GDZRO_64
- SYS$MGBLSC_64
- SYS$DGBLSC

SYS$CREATE_GDZRO と SYS$CRMPSC_GDZRO_64 は新しい状態コードを返します。

状態コード	説明
SS$_INV_SHMEM	共用メモリが不正である。
SS$_INSFRPGS	空き共用ページまたはプライベート・ページが不足している。
SS$_NOBREAK	Galaxy ロックが別のノードによって保有されており，破棄されていない。
SS$_LOCK_TIMEOUT	Galaxy ロック・タイムアウトが発生した。

INSTALL LIST/GLOBAL コマンドと SHOW MEMORY コマンドは Galaxy-wide セクションを認識します。

Galaxy-wide セクションはそれぞれ独自のネーム・スペースを使用します。 さまざまな所有者 UIC に対して，システム・グローバル・セクションとグループ・グローバル・セクションを識別するために常に同じ名前を使用できるのと同様に，Galaxy-wide システム・グローバル・セクションと Galaxy-wide グループ・グローバル・セクションで同じ名前を使用できます。

Galaxy-wide セクションにはそれぞれ，セキュリティ・クラスが割り当てられます。

- GLXSYS_GLOBAL_SECTION
- GLXGRP_GLOBAL_SECTION

これらのセキュリティ・クラスは $GET_SECURITY と $SET_SECURITY システム・サービス，DCL コマンド SET/SHOW SECURITY で使用されます。

これらの新しいセキュリティ・クラスは Galaxy 環境でのみ有効です。 Galaxy 以外のノードでは認識されません。

Galaxy-wide グローバル・セクションが共用インスタンスに存在する場合は，そのセキュリティ属性を検索し，影響を与えることだけが可能です。

次の例は，Galaxy-wide セクションの監査メッセージを示しています。

```
%%%%%%%%%  OPCOM  20-MAR-2002 10:44:43.71  %%%%%%%% (from node GLX1 at 20-MAR-2002 10:44:43.85)
Message from user AUDIT$SERVER on GLX1
Security alarm (SECURITY) on GLX1, system id: 19955
Auditable event:          Object creation
Event information:        global section map request
Event time:               20-MAR-2002 10:44:43.84
PID:                      2040011A
Process name:             ANDY
Username:                 ANDY
Process owner:            [ANDY]
Terminal name:            RTA1:
Image name:               MILKY$DKA100:[ANDY]SHM_MAP.EXE;1
Object class name:        GLXGRP_GLOBAL_SECTION
Object name:              [47]WAY____D99DDB03_0$MY_SECTION
Secondary object name:    <Galaxywide global section>
Access requested:         READ,WRITE
Deaccess key:             8450C610
Status:                   %SYSTEM-S-CREATED, file or section did not exist;
                          has been created
```

“Object name” フィールドに注意してください。ここに表示されているオブジェクト名は，OpenVMS Galaxy 内のセクションを一意に識別します。フィールドは次のとおりです。

[47]	(グループ・グローバル・セクションの場合のみ)セクションの作成者の UIC グループを識別する。
WAY____D99DDB03_0$	共用コミュニティの識別子。
MY_SECTION	ユーザが指定したセクションの名前。

ユーザはセキュリティ・プロファイルを設定または表示するために，要求に対してセクション名とクラスだけを指定できます。UIC は常に現在のプロセスから取得され，コミュニティ識別子はプロセスが実行されているコミュニティから取得されます。

Galaxy-wide システム・グローバル・セクションの出力は，"Object class name" と "Objects name" フィールドでのみ異なります。このタイプのセクションのオブジェクト名には，グループ識別フィールドが含まれません。

オブジェクト・クラス名:	GLXSYS_GLOBAL_SECTION
オブジェクト名:	WAY____D99DDB03_0$SYSTEM_SECTION

セキュリティに関する注意:

Galaxy-wide メモリ・セクションのセキュリティ属性は，それがどのインスタンスで実行されているかとは関係なく，プロセスにとって同一でなければなりません。

この条件は，この共用コミュニティに参加するすべてのインスタンスを，すべてのノードがセキュリティ関連ファイルを共用する同種の OpenVMS Cluster に参加させることで実現できます。

- SYSUAF.DAT, SYSUAFALT.DAT (システム登録ファイル)
- RIGHTSLIST.DAT (ライト・データベース)
- VMS$OBJECTS.DAT (オブジェクト・データベース)

特に，保護の変更を Galaxy-wide セクションに自動的に伝達するには，すべての共用インスタンスで同じ物理ファイル (VMS$OBJECTS.DAT) を使用することが必要です。

インストールしたシステムで，これらのファイルが Galaxy 全体で共用されない場合は，Galaxy-wide 共用セクションの作成者は，セクションが各インスタンスで同じセキュリティ属性を持つようにしなければなりません。このためには，手動の操作が必要です。

第19章　OpenVMS Galaxy デバイス・ドライバ

この章では，OpenVMS Alpha バージョン 7.3 で PCI ドライバに対しての ダイレクト・マップ DMA ウィンドウについて説明します。

19.1 ダイレクト・マップ DMA ウィンドウの変更

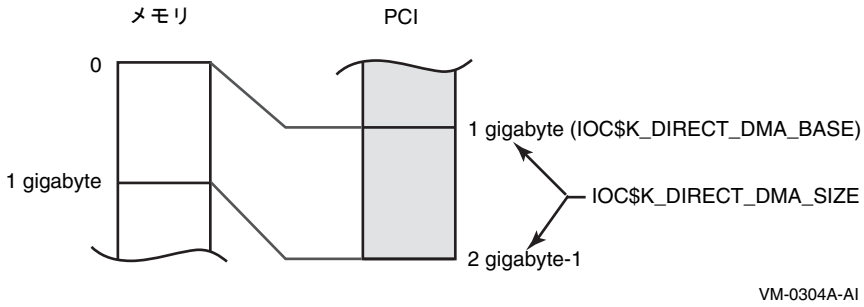
この節で説明する変更は OpenVMS Galaxy とメモリ・ホールをサポートするために OpneVMS バージョン 7.2 で変更されたものです。 この変更では，ダイレクト・マップ DMA ウィンドウを物理メモリ 0 から他の場所に移動します。 この章では，まだ OpenVMS バージョン 7.2 またはそれ以降のドライバにアップデートしていない場合にドライバを更新するのに必要な情報とその背景について，詳しく説明します。

この章では，バス・アドレス可能プール (BAP) については説明しません。

19.2 OpenVMS バージョン 7.2 より以前に PCI ダイレクト・マップ DMA がどのように機能していたか

図 19-1 「PCI ベース DMA」 に示すように，すべての PCI ベースのマシンで， ダイレクト・マップ DMA ウィンドウは PCI 空間の (通常は) 1 GB から始まり， 1 GB の場合は 0 から始まる物理メモリを使用します。

図 19-1 PCI ベース DMA



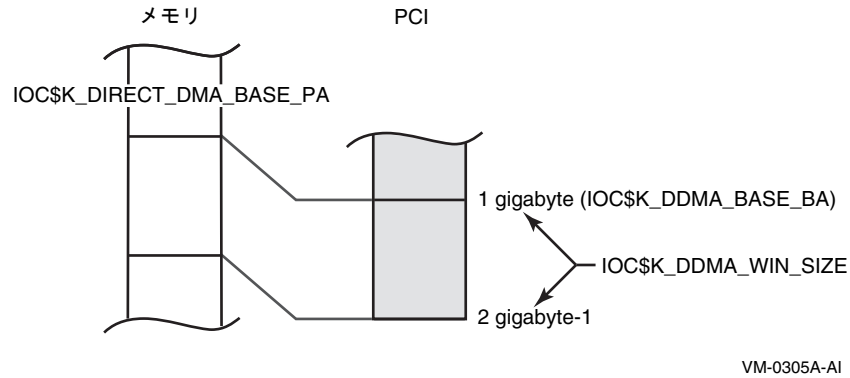
通常，ドライバはそれぞれのバッファ・アドレスを， IOC$K_DIRECT_DMA_SIZE ファンクション・コードを指定して IOC$NODE_DATA を呼び出すことで返されたウィンドウの長さと比較します。 ここでは，メモリ側のウィンドウが 0 から始まるものと仮定しています。 マップ・レジスタが必要かどうかを判断するためによく使われている別の方法として，MMG$GL_MAXPFN を確認する方法があります。 この方法も OpenVMS バージョン 7.3 では正しく動作しない可能性があります。

もっとわかりやすい図と説明については『Writing OpenVMS Alpha Device Drivers in C』を参照してください。

19.3 現在のバージョンの OpenVMS で PCI ダイレクト・マップ DMA がどのように機能するか

図 19-2 「OpenVMS DMA」 に示すように Galaxy およびメモリ・ホールについて考慮すると， ダイレクト・マップ DMA ウィンドウの位置を変更しなければならなくなります。

図 19-2 OpenVMS DMA



メモリ内でダイレクト・マップ DMA ウィンドウのベースがどこにあるかをドライバから判断することはできません。単にバッファ・アドレスをウィンドウの長さと比較しても，バッファがダイレクト・マップ DMA ウィンドウの内部にあるかどうか判断するのに十分ではありません。また，MMG$GL_MAXPFN に対して比較しても， すべてのプールがウィンドウの内部にあるかどうか保証できません。 チェックしなければならない正しいセルは MMG$GL_MAX_NODE_PFN です。 さらに，アライメントについて考慮すると，物理バス・アドレスの計算に少し異なるオフセットを組み込まなければならなくなります。

19.4 0 以外のダイレクト・マップ DMA ウィンドウをサポートするための IOC$NODE_DATA の変更

この問題に対処するために， 新しいファンクション・コードが IOC$NODE_DATA に追加されました。ここではダイレクト・マップ DMA に関連するすべてのコードのリストを示し，データがどのような意味を持つかについても説明します。

コード	説明
IOC$K_DIRECT_DMA_BASE	これは PCI 側のベース・アドレスであるか，またはバス・アドレスである。 これは，IOC$K_DDMA_BASE_BA というファンクション・コードの同意語である。 32 ビットの結果が返される。
IOC$DIRECT_DMA_SIZE	Galaxy 以外のマシンでは， これはダイレクト・マップ DMA ウィンドウのサイズ (M バイト数) を返す。 ダイレクト・マップ DMA ウィンドウが 0 から始まらないシステムでは， 返されるデータは 0 であり， ダイレクト・マップ DMA ウィンドウが存在しないことを示す。 32 ビットの結果が返される。
IOC$K_DDMA_WIN_SIZE	すべてのシステムで， これはダイレクト・マップ DMA ウィンドウのサイズを返す (M バイト単位)。 32 ビットの結果が返される。
IOC$K_DIRECT_DMA_BASE_PA	これはダイレクト・マップ DMA ウィンドウのメモリ内でのベース物理アドレスである。32 ビットの結果が返される。

IOC$K_DIRECT_DMA_BASE_PA コードを使用して返されるアドレスは， オフセットを計算するのに必要です (これは通常，メモリ PA とバス・アドレスの間の 1 GB の差として使用されます)。 オフセットは，ベース・バス・アドレスとベース・メモリ・アドレスの間の符号付きの差として定義されます。これは必ずしも 1 GB ではありません。

付録A OpenVMS Galaxy CPU Load Balancer プログラム

この付録では，特権付きコード・アプリケーションのプログラムの例を示します。このプログラムは，OpenVMS Galaxy 内のインスタンス間で CPU リソースを動的に再割り当てします。

A.1 CPU Load Balancer の概要

OpenVMS Galaxy CPU Load Balancer プログラムは特権を必要とするアプリケーションであり，OpenVMS Galaxy で，インスタンス間で CPU リソースを動的に再割り当てします。

このプログラムは各インスタンスで実行しなければなりません。各イメージは小さな共用メモリ・セクションを作成し (またはこのセクションにマップされ)，そのインスタンスの COM キューの深さに関する情報を定期的に送信します。このデータの平均値をもとに，各インスタンスは最も使用頻度の高いインスタンスと低いインスタンスを判断します。これらの要素が指定された期間に存在する場合は，使用可能なセカンダリ・プロセッサのある使用頻度の最低のインスタンスが，プロセッサの 1 つを使用頻度の最高のインスタンスに再割り当てすることにより，OpenVMS Galaxy でプロセッサの使用のバランス調整が効果的に行われます。このプログラムではコマンド・ライン引数を指定できるので，負荷バランス・アルゴリズムを細かく調整できます。ただし，このプログラムはエラー処理に関して不十分な部分があります。

このプログラムでは，次の OpenVMS Galaxy システム・サービスを使用します。

サービス	説明
SYS$CPU_TRANSITION	CPU の再割り当て
SYS$CRMPSC_GDZRO_64	共用メモリの作成
SYS$SET_SYSTEM_EVENT	OpenVMS Galaxy イベントの通知
SYS$*_GALAXY_LOCK_*	OpenVMS Galaxy ロック

OpenVMS Galaxy リソースは常に，プッシュ・モデルによって再割り当てされ，このモデルでは所有者インスタンスだけがそのリソースを解放できるので，OpenVMS Galaxy の各インスタンスでこのプロセスのコピーを 1 つずつ実行しなければなりません。

このプログラムは OpenVMS バージョン 7.2 以上のマルチインスタンス Galaxy で のみ実行できます。

A.1.1 必要な特権

CPU キューを数えるには，CMKRNL 特権が必要です。共用メモリをマップするには，SHMEM 特権が必要です。

A.1.2 構築とコピーの手順

次の節からの説明に従ってサンプル・プログラムをコンパイルおよびリンクするか，SYS$EXAMPLES:GCU$BALANCER.EXE にあるコンパイル済みのイメージを SYS$COMMON:[SYSEXE]GCU$BALANCER.EXE にコピーします。

OpenVMS Galaxy インスタンスで個別のシステム・ディスクを使用する場合は，各インスタンスに対してここに示した操作を実行する必要があります。

サンプル・プログラムを変更する場合は，次のようにコンパイルとリンクを行ってください。

```
$ CC GCU$BALANCER.C+SYS$LIBRARY:SYS$LIB_C/LIBRARY
$ LINK/SYSEXE GCU$BALANCER
```

A.1.3 スタートアップ・オプション

このプログラムに対して DCL コマンドを設定しなければなりません。この目的でサンプル・コマンド・テーブル・ファイルが提供されます。新しいコマンドをインストールするには，次のように入力します。

```
$ SET COMMAND/TABLE=SYS$LIBRARY:DCLTABLES -
_$ /OUT=SYS$COMMON:[SYSLIB]DCLTABLES GCU$BALANCER.CLD
```

このコマンドは，新しいコマンド定義を共通のシステム・ディレクトリの DCLTABLES.EXE に挿入します。新しいコマンド・テーブルは，システムのリブート時に有効になります。リブートを行わない場合は，次のように入力します。

```
$ INSTALL REPLACE SYS$COMMON:[SYSLIB]DCLTABLES.EXE
```

このコマンドの後，任意のアクティブ・プロセスからコマンドを使用するには，いったんログアウトした後，再びログインする必要があります。 ログアウトしない場合は，Balancer を実行する各プロセスから次のように入力します。

```
$ SET COMMAND GCU$BALANCER.CLD
```

コマンドを設定した後，さまざまなコマンド・ライン・パラメータを使用して， Balancer のアルゴリズムを制御できます。

```
$ CONFIGURE BALANCER{/STATISTICS} x y time
```

このコマンドでは，x は負荷サンプルの数であり，y はリソースの再割り当てを起動するのに必要な，キューに登録されたプロセスの数です。 time は負荷サンプリングの間のデルタ時間です。

/STATISTICS 修飾子を指定すると，状態行に情報が継続的に表示されます。 これはパラメータを調整するのに役立ちます。この出力は，GCU を介して起動した場合など，Balancer が独立プロセスとして実行されている場合は表示されません。 /STATISTICS 修飾子は，DECterm ウィンドウで DCL から直接 Balancer を起動する場合にだけ使用されます。次に例を示します。

```
$ CONFIG BAL 3 1 00:00:05.00
```

これは Balancer を起動して，5 秒ごとにシステムの負荷をサンプリングします。 3 回サンプリングした後，インスタンスの 1 つ以上のプロセスが COM キューに登録されている場合は，リソース (CPU) の再割り当てが行われ， このインスタンスに別の CPU が割り当てられます。

A.2 プログラム例

```
/*
** COPYRIGHT (c) 1998 BY COMPAQ COMPUTER CORPORATION ALL RIGHTS RESERVED.
**
** THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
** ONLY  IN  ACCORDANCE  OF  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE
** INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER
** COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
** OTHER PERSON.  NO TITLE TO AND  OWNERSHIP OF THE  SOFTWARE IS  HEREBY
** TRANSFERRED.
**
** THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE
** AND  SHOULD  NOT  BE  CONSTRUED  AS A COMMITMENT BY COMPAQ COMPUTER
** CORPORATION.
**
** COMPAQ ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS
** SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY COMPAQ OR DIGITAL.
**
**=================================================================
** WARNING - This example is provided for instructional and demo
**           purposes only.  The resulting program should not be
**           run on systems which make use of soft-affinity
**           features of OpenVMS, or while running applications
**           which are tuned for precise processor configurations.
**           We are continuing to explore enhancements such as this
**           program which will be refined and integrated into
**           future releases of OpenVMS.
```

```
**============================================================================
**
** GCU$BALANCER.C - OpenVMS Galaxy CPU Load Balancer.
**
** This is an example of a privileged application which dynamically
** reassigns CPU resources among instances in an OpenVMS Galaxy.  The
** program must be run on each participating instance. Each image will
** create, or map to, a small shared memory section and periodically
** post information regarding the depth of that instances' COM queues.
** Based upon running averages of this data, each instance will
** determine the most, and least busy instance.  If these factors
** exist for a specified duration, the least busy instance having
** available secondary processors, will reassign one of its processors
** to the most busy instance, thereby effectively balancing processor
** usage across the OpenVMS Galaxy.  The program provides command line
** arguments to allow tuning of the load balancing algorithm.
** The program is admittedly shy on error handling.
**
** This program uses the following OpenVMS Galaxy system services:
**
**      SYS$CPU_TRANSITION   - CPU reassignment
**      SYS$CRMPSC_GDZRO_64  - Shared memory creation
**      SYS$SET_SYSTEM_EVENT - OpenVMS Galaxy event notification
**      SYS$*_GALAXY_LOCK_*  - OpenVMS Galaxy locking
**
** Since OpenVMS Galaxy resources are always reassigned via a "push"
** model, where only the owner instance can release its resources,
** one copy of this process must run on each instance in the OpenVMS
** Galaxy.
**
** ENVIRONMENT: OpenVMS V7.2 Multiple-instance Galaxy.
**
** REQUIRED PRIVILEGES:  CMKRNL required to count CPU queues
**                       SHMEM  required to map shared memory
**
** BUILD/COPY INSTRUCTIONS:
**
** Compile and link the example program as described below, or copy the
** precompiled image found in SYS$EXAMPLES:GCU$BALANCER.EXE to
** SYS$COMMON:[SYSEXE]GCU$BALANCER.EXE
**
** If your OpenVMS Galaxy instances utilize individual system disks,
** you will need to do the above for each instance.
**
** If you change the example program, compile and link it as follows:
**
**   $ CC GCU$BALANCER.C+SYS$LIBRARY:SYS$LIB_C/LIBRARY
**   $ LINK/SYSEXE GCU$BALANCER
**
** STARTUP OPTIONS:
**
** You must establish a DCL command for this program.  We have provided a
** sample command table file for this purpose. To install the new
** command, do the following:
**
**    $ SET COMMAND/TABLE=SYS$LIBRARY:DCLTABLES -
**      /OUT=SYS$COMMON:[SYSLIB]DCLTABLES GCU$BALANCER.CLD
**
** This command inserts the new command definition into DCLTABLES.EXE
** in your common system directory.  The new command tables will take
** effect when the system is rebooted.  If you would like to avoid a
** reboot, do the following:
**
**    $ INSTALL REPLACE SYS$COMMON:[SYSLIB]DCLTABLES.EXE
**
```

```
** After this command, you will need to log out, then log back in to
** use the command from any active processes.  Alternatively, if you
** would like to avoid logging out, do the following from each process
** you would like to run the balancer from:
**
**    $ SET COMMAND GCU$BALANCER.CLD
**
** Once your command has been established, you may use the various
** command line parameters to control the balancer algorithm.
**
**    $ CONFIGURE BALANCER{/STATISTICS} x y time
**
** Where: "x" is the number of load samples to take.
**        "y" is the number of queued processes required to trigger
**            resource reassignment.
**        "time" is the delta time between load sampling.
**
** The /STATISTICS qualifier causes the program to display a
** continuous status line.  This is useful for tuning the parameters.
** This output is not visible if the balancer is run detached, as is
** the case if it is invoked via the GCU.  It is intended to be used
** only when the balancer is invoked directly from DCL in a DECterm
** window.
**
** For example: $ CONFIG BAL 3 1 00:00:05.00
**
**        Starts the balancer which samples the system load every
**        5 seconds.  After 3 samples, if the instance has one or
**        more processes in the COM queue, a resource (CPU)
**        reassignment will occur, giving this instance another CPU.
**
** GCU STARTUP:
**
** The GCU provides a menu item for launching SYS$SYSTEM:GCU$BALANCER.EXE
** and a dialog for altering the balancer algorithm.  These features will
** only work if the balancer image is properly installed as described
** the the following paragraphs.
**
** To use the GCU-resident balancer startup option, you must:
**
** 1) Compile, link, or copy the balancer image as described previously.
** 2) Invoke the GCU via: $ CONFIGURE GALAXY   You may need to set your
**    DECwindows display to a suitably configured workstation or PC.
** 3) Select the "CPU Balancer" entry from the "Galaxy" menu.
** 4) Select appropriate values for your system.  This may take some
**    testing.  By default, the values are set aggressively so that
**    the balancer action can be readily observed.  If your system is
**    very heavily loaded, you will need to increase the values
**    accordingly to avoid excessive resource reassignment.  The GCU
**    does not currently save these values, so you may want to write
**    them down once you are satisfied.
** 5) Select the instance/s you wish to have participate, then select
**    the "Start" function, then press OK.  The GCU should launch the
**    process GCU$BALANCER on all selected instances.  You may want to
**    verify these processes have been started.
**
** SHUTDOWN WARNING:
**
** In an OpenVMS Galaxy, no process may have shared memory mapped on an
** instance when it leaves the Galaxy, as during a shutdown. Because of
** this, SYS$MANAGER:SYSHUTDWN.COM must be modified to stop the process
** if the GCU$BALANCER program is run from a SYSTEM UIC.  Processes in the
** SYSTEM UIC group are not terminated by SHUTDOWN.COM when shutting down
** or rebooting OpenVMS. If a process still has shared memory mapped when
** an instance leaves the Galaxy, the instance will crash with a
```

```
** GLXSHUTSHMEM bugcheck.
**
** To make this work, SYS$MANAGER:SYSHUTDWN.COM must stop the process as
** shown in the example below.  Alternatively, the process can be run
** under a suitably privileged, non-SYSTEM UIC.
**
** SYSHUTDWN.COM EXAMPLE - Paste into SYS$MANAGER:SYSHUTDWN.COM
**
**    $!
**    $! If the GCU$BALANCER image is running, stop it to release shmem.
**    $!
**    $ procctx = f$context("process",ctx,"prcnam","GCU$BALANCER","eql")
**    $ procid  = f$pid(ctx)
**    $ if procid .NES. "" then $ stop/id='procid'
**
** Note, you could also just do a "$ STOP GCU$BALANCER" statement.
**
** OUTPUTS:
**
**    If the logical name GCU$BALANCER_VERIFY is defined, notify the
**    SYSTEM account when CPUs are reassigned.  If the /STATISTICS
**    qualifier is specified, a status line is continually displayed,
**    but only when run directly from the command line.
**
** REVISION HISTORY:
**
** 02-Dec-1998 Greatly improved instructions.
** 03-Nov-1998 Improved instructions.
** 24-Sep-1998 Initial code example and integration with GCU.
*/
#include <brkdef>
#include <builtins>
#include <cstdef>
#include <descrip>
#include <glockdef>
#include <ints>
#include <pdscdef>
#include <psldef>
#include <secdef>
#include <ssdef>
#include <starlet>
#include <stdio>
#include <stdlib>
#include <string>
#include <syidef>
#include <sysevtdef>
#include <vadef>
#include <vms_macros>
#include <cpudef>
#include <iosbdef.h>
#include <efndef.h>
/* For CLI */
#include <cli$routines.h>
#include <chfdef.h>
#include <climsgdef.h>

#define HEARTBEAT_RESTART     0 /* Flags for synchronization           */
#define HEARTBEAT_ALIVE       1
#define HEARTBEAT_TRANSPLANT  2

#define GLOCK_TIMEOUT    100000 /* Sanity check, max time holding gLock */
#define _failed(x) (!( (x) & 1) )

$DESCRIPTOR(system_dsc, "SYSTEM");       /* Brkthru account name   */
$DESCRIPTOR(gblsec_dsc, "GCU$BALANCER"); /* Global section name    */
```

```
struct  SYI_ITEM_LIST {              /* $GETSYI item list format */
  short buflen,item;
  void *buffer,*length;
};

/* System information and an item list to use with $GETSYI */

static unsigned long total_cpus;
static uint64   partition_id;
static long     max_instances = 32;
iosb            g_iosb;

struct SYI_ITEM_LIST syi_itemlist[3] = {
     {sizeof (long), SYI$_ACTIVECPU_CNT,&total_cpus,  0},
     {sizeof (long), SYI$_PARTITION_ID, &partition_id,0},
     {0,0,0,0}};

extern uint32 *SCH$AQ_COMH;          /* Scheduler COM queue address */
unsigned long PAGESIZE;              /* Alpha page size             */
uint64        glock_table_handle;    /* Galaxy lock table handle    */

/*
** Shared Memory layout (64-bit words):
** ====================================
** 0  to  n-1:  Busy count, where 100 = 1 process in a CPU queue
** n  to  2n-1: Heartbeat (status) for each instance
** 2n to  3n-1: Current CPU count on each instance
** 3n to  4n-1: Galaxy lock handles for modifying heartbeats
**
** where n = max_instances * sizeof(long).
**
** We assume the entire table (easily) fits in two Alpha pages.
*/

/* Shared memory pointers must be declared volatile */

volatile uint64  gs_va = 0;          /* Shmem section address     */
volatile uint64  gs_length = 0;      /* Shmem section length      */
volatile uint64 *gLocks;             /* Pointers to gLock handles */
volatile uint64 *busycnt,*heartbeat,*cpucount;

/*****************************************************************/
/* FUNCTION init_lock_tables - Map to the Galaxy locking table and   */
/* create locks if needed. Place the lock handles in a shared memory */
/* region, so all processes can access the locks.                    */
/*                                                                   */
/* ENVIRONMENT: Requires SHMEM and CMKRNL to create tables.          */
/* INPUTS:      None.                                                */
/* OUTPUTS:     Any errors from lock table creation.                 */
/*****************************************************************/
int init_lock_tables (void)
{
    int status,i;
    unsigned long sanity;
    uint64 handle;
    unsigned int min_size, max_size;

/* Lock table names are 15-byte padded values, unique across a Galaxy.*/
   char table_name[] = "GCU_BAL_GLOCK  ";

/* Lock names are 15-byte padded values, but need not be unique. */
   char lock_name[] = "GCU_BAL_LOCK   ";

/* Get the size of a Galaxy lock */
```

```
 status = sys$get_galaxy_lock_size(&min_size,&max_size);
 if (_failed(status) ) return (status);

    /*
    ** Create or map to a process space Galaxy lock table. We assume
    ** one page is enough to hold the locks. This will work for up
    ** to 128 instances.
    */
    status = sys$create_galaxy_lock_table(table_name,PSL$C_USER,
                PAGESIZE,GLCKTBL$C_PROCESS,0,min_size,&glock_table_handle);
    if (_failed(status) ) return (status);

    /*
    ** Success case 1: SS$_CREATED
    ** We created the table, so  populate it with locks and
    ** write the handles to shared memory so the other partitions
    ** can access them. Only one instance can receive SS$_CREATED
    ** for a given lock table; all other mappers will get SS$_NORMAL.
    */
    if (status == SS$_CREATED)
    {
      printf ("%%GCU$BALANCER-I-CRELOCK, Creating G-locks\n");
      for (i=0; i<max_instances>pdsc$q_entry[0];
   sub_addr[1] = sub_addr[0] + PAGESIZE;
   if (__PAL_PROBER( (void *)sub_addr[0],sizeof(int),PSL$C_USER) != 0)
        sub_addr[1] = sub_addr[0];

   status = sys$lkwset(sub_addr,locked_code,PSL$C_USER);
   if (_failed(status) ) exit(status);
}

/*****************************************************************/
/* FUNCTION reassign_a_cpu - Reassign a single CPU to another    */
/* instance.                                                     */
/*                                                               */
/* ENVIRONMENT: Requires CMKRNL privilege.                       */
/* INPUTS:      most_busy_id: partition ID of destination.       */
/* OUTPUTS:     None.                                            */
/*                                                               */
/* Donate one CPU at a time - then wait for the remote instance to   */
/* reset its heartbeat and recalculate its load.                 */
/*****************************************************************/
void reassign_a_cpu(int most_busy_id)
{
  int status,i;
  static char op_msg[255];
  static char iname_msg[1];
  $DESCRIPTOR(op_dsc,op_msg);
  $DESCRIPTOR(iname_dsc,"");
  iname_dsc.dsc$w_length = 0;

  /* Update CPU info */

  status = sys$getsyiw(EFN$C_ENF,0,0,&syi_itemlist, &g_iosb,0,0);
  if (_failed(status) ) exit(status);

  /* Don't attempt reassignment if we are down to one CPU */

  if (total_cpus > 1)
  {
  status = sys$acquire_galaxy_lock(gLocks[most_busy_id],GLOCK_TIMEOUT,0);
  if (_failed(status) ) exit(status);
  heartbeat[most_busy_id] = HEARTBEAT_TRANSPLANT;
  status = sys$release_galaxy_lock(gLocks[most_busy_id]);
  if (_failed(status) ) exit(status);
```

```
  status = sys$cpu_transitionw(CST$K_CPU_MIGRATE,CST$K_ANY_CPU,0,
                               most_busy_id,0,0,0,0,0,0);
  if (status & 1)
  {
    if (getenv ("GCU$BALANCER_VERIFY") )
    {
      sprintf(op_msg,
             "\n\n*****GCU$BALANCER: Reassigned a CPU to instance %li\n",
             most_busy_id);
      op_dsc.dsc$w_length = strlen(op_msg);
      sys$brkthru(0,&op_dsc,&system_dsc,BRK$C_USERNAME,0,0,0,0,0,0,0);
      }
      update_cpucount(0);  /* Update the CPU count after donating one */
    }
  }
}

/*****************************************************************/
/* IMAGE ENTRY - MAIN                                            */
/*                                                               */
/* ENVIRONMENT: OpenVMS Galaxy                                   */
/* INPUTS:      None.                                            */
/* OUTPUTS:     None.                                            */
/*****************************************************************/
int main(int argc, char **argv)
{
   int           show_stats = 0;
   long          busy,most_busy,nprocs;
   int64         delta;
   unsigned long status,i,j,k,system_cpus,instances;
   unsigned long arglst         = 0;
   uint64        version_id[2]  = {0,1};
   uint64        region_id      = VA$C_P0;
   uint64        most_busy_id,cpu_hndl = 0;

/* Static descriptors for storing parameters.  Must match CLD defs */

  $DESCRIPTOR(p1_desc,"P1");
  $DESCRIPTOR(p2_desc,"P2");
  $DESCRIPTOR(p3_desc,"P3");
  $DESCRIPTOR(p4_desc,"P4");
  $DESCRIPTOR(stat_desc,"STATISTICS");

/* Dynamic descriptors for retrieving parameter values */

  struct dsc$descriptor_d samp_desc = {0,DSC$K_DTYPE_T,DSC$K_CLASS_D,0};
  struct dsc$descriptor_d proc_desc = {0,DSC$K_DTYPE_T,DSC$K_CLASS_D,0};
  struct dsc$descriptor_d time_desc = {0,DSC$K_DTYPE_T,DSC$K_CLASS_D,0};

  struct SYI_ITEM_LIST syi_pagesize_list[3] = {
    {sizeof (long), SYI$_PAGE_SIZE      , &PAGESIZE    ,0},
    {sizeof (long), SYI$_GLX_MAX_MEMBERS,&max_instances,0},
    {0,0,0,0}};
/*
** num_samples and time_desc determine how often the balancer should
** check to see if any other instance needs more CPUs. num_samples
** determines the number of samples used to calculate the running
** average, and sleep_dsc determines the amount of time between
** samples.
**
** For example, a sleep_dsc of 30 seconds and a num_samples of 20 means
** that a running average over the last 10 minutes (20 samples * 30 secs)
** is used to balance CPUs.
**
```

```
** load_tolerance is the minimum load difference which triggers a CPU
** migration. 100 is equal to 1 process in the computable CPU queue.
*/
   int num_samples;     /* Number of samples in running average      */
   int load_tolerance;  /* Minimum load diff to trigger reassignment */

/* Parse the CLI */
                                                  /* CONFIGURE VERB */
   status      = CLI$PRESENT(&p1_desc);           /* BALANCER       */
   if (status != CLI$_PRESENT) exit(status);
   status      = CLI$PRESENT(&p2_desc);           /* SAMPLES        */
   if (status != CLI$_PRESENT) exit(status);
   status      = CLI$PRESENT(&p3_desc);           /* PROCESSES      */
   if (status != CLI$_PRESENT) exit(status);
   status      = CLI$PRESENT(&p4_desc);           /* TIME           */
   if (status != CLI$_PRESENT) exit(status);

   status     = CLI$GET_VALUE(&p2_desc,&samp_desc);
   if (_failed(status) ) exit(status);
   status     = CLI$GET_VALUE(&p3_desc,&proc_desc);
   if (_failed(status) ) exit(status);
   status     = CLI$GET_VALUE(&p4_desc,&time_desc);
   if (_failed(status) ) exit(status);
   status     = CLI$PRESENT(&stat_desc);
   show_stats = (status == CLI$_PRESENT) ? 1 : 0;

   num_samples = atoi(samp_desc.dsc$a_pointer);
   if (num_samples <= 0) num_samples = 3;

   load_tolerance = (100 * (atoi(proc_desc.dsc$a_pointer) ) );
   if (load_tolerance <= 0) load_tolerance = 100;

   if (show_stats)
     printf("Args: Samples: %d, Processes: %d, Time: %s\n",
        num_samples,load_tolerance/100,time_desc.dsc$a_pointer);

   lockdown();                   /* Lock down the cpu_q subroutine */

   /* Get the page size and max members for this system */

   status = sys$getsyiw(EFN$C_ENF,0,0,&syi_pagesize_list,&g_iosb,0,0);
   if (_failed(status) ) return (status);

   if (max_instances == 0) max_instances = 1;

   /* Get our partition ID and initial CPU info */

   status = sys$getsyiw(EFN$C_ENF,0,0,&syi_itemlist,&g_iosb,0,0);
   if (_failed(status) ) return (status);

   /* Map two pages of shared memory */

status = sys$crmpsc_gdzro_64(&gblsec_desc,version_id,0,PAGESIZE+PAGESIZE,
         &region_id,0,PSL$C_USER,(SEC$M_EXPREG|SEC$M_SYSGBL|SEC$M_SHMGS),
         &gs_va,&gs_length);
if (_failed(status) ) exit(status);

   /* Initialize the pointers into shared memory */

   busycnt   = (uint64 *) gs_va;
   heartbeat = (uint64 *) gs_va     + max_instances;
   cpucount  = (uint64 *) heartbeat + max_instances;
   gLocks    = (uint64 *) cpucount  + max_instances;

   cpucount[partition_id] = total_cpus;
```

```
    /* Create or map the Galaxy lock table */

    status = init_lock_tables();
    if (_failed(status) ) exit(status);

    /* Initialize delta time for sleeping */

    status = sys$bintim(&time_desc,&delta);
    if (_failed(status) ) exit(status);

    /*
    ** Register for CPU migration events. Whenever a CPU is added to
    ** our active set, the routine "update_cpucount" will fire.
    */
    status = sys$set_system_event(SYSEVT$C_ADD_ACTIVE_CPU,
                update_cpucount,0,0,SYSEVT$M_REPEAT_NOTIFY,&cpu_hndl);
    if (_failed(status) ) exit(status);

    /* Force everyone to resync before we do anything */

    for (j=0; j<max_instances; j++)
    {
      status = sys$acquire_galaxy_lock(gLocks[j],GLOCK_TIMEOUT,0);
      if (_failed(status) ) exit(status);
      heartbeat[j] = HEARTBEAT_RESTART;
      status = sys$release_galaxy_lock (gLocks[j]);
      if (_failed(status) ) exit(status);
    }

    printf("%%GCU$BALANCER-S-INIT, CPU balancer initialized.\n\n");

    /*** Main loop ***/
    do
    {
      /* Calculate a running average and update it */

      nprocs = sys$cmkrnl(cpu_q,&arglst) * 100;

/* Check out our state... */

switch (heartbeat[partition_id])
{
  case HEARTBEAT_RESTART: /* Mark ourself for reinitializition. */
  {
    update_cpucount(0);
    status = sys$acquire_galaxy_lock(gLocks[partition_id],GLOCK_TIMEOUT,0);
    if (_failed(status) ) exit(status);
    heartbeat[partition_id] = HEARTBEAT_ALIVE;
    status = sys$release_galaxy_lock(gLocks[partition_id]);
    if (_failed(status) ) exit(status);
    break;
  }
  case HEARTBEAT_ALIVE: /* Update running average and continue. */
  {
    busy = (busycnt[partition_id]*(num_samples-1)+nprocs)/num_samples;
    busycnt[partition_id] = busy;
    break;
   }
   case HEARTBEAT_TRANSPLANT:  /* Waiting for a new CPU to arrive. */
   {
    /*
     ** Someone just either reset us, or gave us a CPU and put a wait
     ** on further donations.  Reassure the Galaxy that we're alive,
     ** and calculate a new busy count.
```

```
       */
      busycnt[partition_id] = nprocs;
      status = sys$acquire_galaxy_lock(gLocks[partition_id],GLOCK_TIMEOUT,0);
      if (_failed(status) ) exit(status);
      heartbeat[partition_id] = HEARTBEAT_ALIVE;
      status = sys$release_galaxy_lock(gLocks[partition_id]);
      if (_failed(status) ) exit(status);
      break;
     }
     default:         /* This should never happen. */
     {
       exit(0);
       break;
     }
    }

    /* Determine the most_busy instance. */

    for (most_busy_id=most_busy=i=0; i<max_instances; i++)
    {
      if (busycnt[i] > most_busy)
      {
        most_busy_id = (uint64) i;
        most_busy    = busycnt[i];
      }
    }

    if (show_stats)
      printf("Current Load: %3Ld, Busiest Instance: %Ld, Queue Depth: %4d\r",
            busycnt[partition_id],most_busy_id,(nprocs/100) );

    /* If someone needs a CPU and we have an extra, donate it. */

     if ( (most_busy > busy + load_tolerance) &&
         (cpucount[partition_id] > 1) &&
         (heartbeat[most_busy_id] != HEARTBEAT_TRANSPLANT) &&
         (most_busy_id != partition_id) )
       {
          reassign_a_cpu(most_busy_id);
       }

       /* Hibernate for a while and do it all again. */

       status = sys$schdwk(0,0,&delta,0);
       if (_failed(status) ) exit(status);
       status = sys$hiber();
       if (_failed(status) ) exit(status);

     } while (1);
     return (1);
}
```

付録B　メモリ・サイズ設定の共通値

表 B-1 「メモリ・サイズ環境変数の共通値」 は，メモリ・サイズを設定するために使用する Galaxy 環境変数の共通値を示しています。 すべての値は，16 進バイトで表記されています。

表 B-1 メモリ・サイズ環境変数の共通値

1 MB	0x 10 0000
2 MB	0x 20 0000
4 MB	0x 40 0000
8 MB	0x 80 0000
16 MB	0x 100 0000
32 MB	0x 200 0000
64 MB	0x 400 0000
128 MB	0x 800 0000
256 MB	0x 1000 0000
448 MB	0x1C00 0000
512 MB	0x 2000 0000
1 GB	0x 4000 0000
2 GB	0x 8000 0000
4 GB	0x 1 0000 0000
8 GB	0x 2 0000 0000
16 GB	0x 4 0000 0000
32 GB	0x 8 0000 0000
64 GB	0x 10 0000 0000
128 GB	0x 20 0000 0000
256 GB	0x 40 0000 0000
512 GB	0x 80 0000 0000
1 TB	0x 100 0000 0000

付録C ライセンスのインストール

この付録では，ハード・パーティションおよびソフト・パーティションでライセンス・ユニットを共用する方法を説明します。

C.1 ライセンス・プロセス

共通ライセンス・データベースで，ハード・パーティションとソフト・パーティション間でのライセンス・ユニットの共用ができるように (サポートされている場合)，次の手順を実行します。

1. 必要なユニット数を算出します。
   - OpenVMS 基本ライセンスをロードします。
   - SMP ライセンスをロードします。
   - SHOW LICENSE /UNIT_REQUIREMENTS /CLUSTER コマンドを使って，正しい数のライセンス・ユニットを持っていることを確認します。

注意:

会話型でログインするには，SMP のライセンスが 1 つ必要です。 OpenVMS 基本ライセンスでは会話型のログインを行うことはできません。

---

2. ライセンスを共通ライセンス・データベースに追加します。 たとえば，次のようにします。

```
$ LICENSE REGISTER OPENVMS-ALPHA /ISSUER=DEC -
_$ /AUTHORIZATION=USA123456 -
_$ /PRODUCER=DEC -
_$ /UNITS=1050 -
_$ /AVAILABILITY=H -
_$ /OPTIONS=(N0_SHARE) -
_$ /CHECKSUM=2-BGON-IAMA-GNOL-AIKO
```

注意:

LICENSE REGISTER コマンドに /INCLUDE 修飾子を指定して，ライセンスの NO_SHARE 属性より優先させることはできません。

---

3. LICENSE MODIFY /INCLUDE=(nodename-list) コマンドを使用してライセンスを変更し，PAK の NO_SHARE 属性より優先させます。 たとえば，次のようにします。

```
$ LICENSE MODIFY OPENVMS-ALPHA -
_$ /AUTHORIZATION=USA123456 -
_$ /INCLUDE=(NODEA, NODEb, NODEC)
```

4. 各パーティションで動作している OpenVMS インスタンスで OpenVMS Alpha ライセンス・ユニットを利用できるようにするには，各パーティションで MBM の環境変数 SYS_SERIAL_NUM を同じにしなければなりません。 それには，次の手順を実行します。
   a. MBM コンソールから，SHOW SYS_SERIAL_NUM コマンドを使用してシステムのシリアル番号を表示します。 たとえば次のようにします。

```
MBM>>> SHOW SYS_SERIAL_NUM
sys_serial_num G2A105
```

   b. SYS_SERIAL_NUM の値が空白の場合は，MBM コンソールで SET SYS_SERIAL_NUM コマンドを使用して，システム・シリアル番号を設定します。システムのシリアル番号はすべてのパーティション (ハードとソフト) に通知されます。 たとえば，次のようにします。

```
MBM>>> SET SYS_SERIAL_NUM G2A105
```

注意:

ゼロ以外の値を最大 12 文字で入力する必要があります (以前のシステムでは 16 文字までサポートされていました)。 作成するシステム・シリアル番号が，この OpenVMS Cluster の他の AlphaServer システムで使われていないことを確認してください。

5. OpenVMS Cluster のライセンス・データベースを正しく更新するために，OpenVMS クラスタのすべての共通ノードを完全にシャットダウンしてリブートすることをお勧めします。ローリング・アップグレードでは，共通ライセンス・データベースが正しく更新されません。

パーティション間で NO_SHARE ライセンスを共有するパーティション対応システムでは，システムのブート時に次のメッセージが表示されることがあります。

```
%LICENSE-E-NOAUTH, DEC OPENVMS-ALPHA use is not authorized on this node
-LICENSE-F-EXCEEDED, attempted usage exceeds active license limits
-LICENSE-I-SYSMGR, please see your system manager
Startup processing continuing...
```

このメッセージは無視して構いません。 OPENVMS-ALPHA PAK をすでに共有しているシステムにログオンすると，このメッセージが表示されます。この状況は，将来のリリースで修正される予定です。

C.1.1 ライセンスについての詳細情報

ライセンスのインストールと管理についての詳細は，『OpenVMS License Management Utility Manual』を参照してください。

用語集

G

Galaxy OpenVMS におけるソフト・パーティションの実装。

Galaxy 以外のシステム (non-Galaxy system) Galaxy を実行していないシステム，または Galaxy を有効にしていないシステム。

あ

インスタンス (instance) 動作中のオペレーティング・システム。

か

コミュニティ (community) メモリなどのリソースの共用を可能にするソフトウェア要素。

構成ツリー (configuration tree) システム・コンポーネントを，ツリー形式の図で表現したもの。

さ

スピンロック (spinlock) ビットロックの状態を繰り返しチェックすること。

ソフト・パーティション (soft partition) ソフトウェア制御のアクセス・バリアにより分離されたコンピューティング・リソースの集まり。

な

ノード (node) 構成ツリー内の接続ポイント。

は

ハード・パーティション (hard partition) ハードウェアで強制されるアクセス・バリアで物理的に分離されたコンピューティング・リソースの集まり。

ま

ミューテクス (mutex) リソース・セマフォのこと。

索引

A
AlphaServer 4100 システム
  OpenVMS Galaxy の構築, 75
AlphaServer 8200 システム
  OpenVMS Galaxy の構築, 69
AlphaServer 8400 システム
  OpenVMS Galaxy の構築, 59
AlphaServer ES40 システム
  OpenVMS Galaxy の構築, 81
AlphaServer ES47 システム
  OpenVMS Galaxy の構築, 93
AlphaServer ES80 システム
  OpenVMS Galaxy の構築, 93
AlphaServer GS1280 システム
  OpenVMS Galaxy の構築, 93
AlphaServer GS160 システム
  OpenVMS Galaxy の構築, 87, 93

C
CD ドライブの推奨, 41
CPU Load Balancer プログラム
  例, 178
CPU Load Balancer プログラムの概要
  構築とコピーの手順, 177
  スタートアップ・オプション, 177
  必要な特権, 177
CPU 再割り当て
  DCL, 153
  Galaxy サービスを使用したソフトウェアによる再割り当て, 154
  GCU のドラッグ・アンド・ドロップ, 153
  インターモダル, 153
  失敗, 154
CPU リソースの管理, 153

D
DCL コマンド
  CONFIGURE, 164
  CONFIGURE GALAXY, 158
  INITIALIZE, 158
  INSTALL, 163
  INSTALL ADD, 164
  INSTALL/LIST, 158
  SET CPU/AUTO_START, 157
  SET CPU/MIGRATE, 157
  SET CPU/OVERRIDE_CHECKS, 157
  SET CPU/POWER, 157
  SET CPU/REFRESH, 157
  SET CPU/START, 157
  SET ENTRY, 158
  SET PROCESS, 53, 158
  SET QUEUE, 158
  SHOW CPU/ACTIVE_SET, 157
  SHOW CPU/CONFIGURE_SET, 157
  SHOW CPU/POTENTIAL_SET, 157
  SHOW CPU/STANDBY_SET, 158
  SHOW CPU/SYSTEM, 158
  SHOW MEMORY, 158, 163
  SHOW PROCESS, 53, 158
  START QUEUE, 158
  START/CPU/POWER, 158
  STOP/CPU/MIGRATE, 158
  STOP/CPU/POWER=OFF, 158
  SUBMIT, 158
DCL コマンドで OpenVMS Galaxy の管理に便利なもの
  CPU コマンド, 161
    SET CPU, 162
    SHOW CPU, 161
    STOP/CPU/MIGRATE, 161
DCL レキシカル
  F$GETJPI, 53, 160
  F$GETSYI, 53

E
ES45, 105

G
Galaxy, 30
Galaxy-wide グローバル・セクション, 173
Galaxy Configuration ユーティリティ (参照 GCU (Galaxy Configuration ユーティリティ))
Galaxy 構成モデル
  アクティブ・モデル, 115
  オフライン・モデル, 115
  オンライン・モデルとオフライン・モデル, 123
Galaxy ソフトウェア・アーキテクチャ, 33
Galaxy に関するヒントと手法
  Galaxy モードのオフ設定, 108
  コンソール環境変数の変更, 107
  コンソールに関するヒント, 107
  システム・オート・アクション, 107
Galaxy の機能, 35
  CPU の再割り当て, 36
  SMP, 35
  クラスタ, 35
  動的な再構成, 36
Galaxy の構成上の考慮点, 40
  EISA バスのサポート, 41
  XMI バスのサポート, 41
  メモリ分割に関する制限事項, 41
Galaxy の利点, 36
  可用性, 36
  互換性, 36
  所有コスト, 37
  スケーラビリティ, 37
  性能, 37
  適応性, 37
Galaxy バージョン 7.3 の機能, 37
Galaxy ハードウェアおよびソフトウェア・コンポーネント
  CPU, 33

I/O, 33
共用メモリ, 33
コンソール, 33
独立インスタンス, 34
GCM, 22
GCU (Galaxy Configuration ユーティリティ)
定義, 109
GCU システム・メッセージ, 123
GCU 操作
Galaxy 構成モデルの作成, 110
会話, 112
監視, 111
GCU チャート
Failover Target チャート, 120
Logical Structure チャート, 117
Memory Assignment チャート, 118
Physical Structure チャート, 117
使用, 116
GCU による OpenVMS Galaxy の管理
独立インスタンス, 113
必要な PROXY アクセス, 113
分離されたインスタンス, 113
GCU メニューのカスタマイズ, 121

H
HP Availability Manager による OpenVMS Galaxy の監視, 122

I
IOC$NODE_DATA の変更, 176

L
LAN (ローカル・エリア・ネットワーク), 167

O
OpenVMS Alpha のインストール
AlphaServer 4100, 76
AlphaServer 8200, 70
AlphaServer 8400, 63
AlphaServer ES40, 83
AlphaSever GS160, 87, 93
OpenVMS Galaxy デバイス・ドライバ, 175
OpenVMS Galaxy の構築
AlphaServer 4100, 75
AlphaServer 8200, 69
AlphaServer 8400, 59
AlphaServer ES40, 81
AlphaServer ES80, 93
AlphaServer GS1280, 93
AlphaServer GS160, 87, 93
AlphaServer GS320, 87, 93
AlphaServer GS80, 87, 93
OpenVMS Galaxy のブート
AlphaServer 8200, 73
AlphaServer 8400, 67

P
PCI ダイレクト・マップ DMA の機能のしくみ
OpenVMS バージョン 7.3 以前の場合, 175
OpenVMS バージョン 7.3 の場合, 175

S
SDA (System Dump Analyzer)
共用メモリのダンプ, 45
の使用, 45
SHOW RAD コマンド, 53
SMCI (共用メモリ・クラスタ・インターコネクト), 167
SRM, 93
System Dump Analyzer (参照 SDA (System Dump Analyzer))

い
移動, 22

お
親, 20

か
可能な構成
shared-everything コンピューティング・モデル, 40
shared-nothing コンピューティング・モデル, 39
shared-partial コンピューティング・モデル, 39
環境変数の設定
AlphaServer 4100, 77
AlphaServer 8200, 71
AlphaServer 8400, 64
AlphaServer ES40, 84
AlphaServer GS160, 88

き
共用メモリ・クラスタ・インターコネクト (参照 SMCI (共用メモリ・クラスタ・インターコネクト))
共用メモリでの通信
LAN (ローカル・エリア・ネットワーク) , 169
SMCI (共用メモリ・クラスタ・インターコネクト), 167
共用メモリの使用, 171
共用メモリ・プログラミング・インタフェース, 171

く
クラスタ情報, 41
Galaxy インスタンスになる, 42
SCSI に関する留意事項, 42

け
兄弟, 20

こ
子, 20
高可用性アプリケーション, 38
構成
ツリー, 20
構成要件
AlphaServer 4100, 75
AlphaServer 8200, 69
AlphaServer 8400, 59
AlphaServer ES40, 81
AlphaServer GS1280, 93

AlphaServer GS160, 87, 93

し
システム・サービス
$ACQUIRE_GALAXY_LOCK, 44
$CLEAR_SYSTEM_EVENT, 44
$CPU_TRANSITION, 154
$CREATE_GALAXY_LOCK, 44
$CREATE_GALAXY_LOCK_TABLE, 44
$CREATE_GDZRO, 53
$CREPRC, 53
$CRMPSC_GDZRO_64, 53
$DELETE_GALAXY_LOCK, 44
$DELETE_GALAXY_LOCK_TABLE, 44
$GET_GALAXY_LOCK_INFO, 44
$GET_GALAXY_LOCK_SIZE, 44
$GETJPI, 53
$GETSYI, 53
$RELEASE_GALAXY_LOCK, 44
$SET_PROCESS_PROPERTIESW, 53
$SET_SYSTEM_EVENT, 44
新しいセクション・フラグ, 172
強化されたサービス, 172
システム・ディスクの作成
AlphaServer 8200, 70
AlphaServer 8400, 63
AlphaServer GS160, 87, 93
システムの初期化とコンソール装置の起動
AlphaServer 4100, 77
AlphaServer ES40, 85
所有関係, 20
シングル・インスタンス Galaxy
定義, 40
任意の Alpha システムを使用した, 105

せ
セカンダリ・コンソール装置の起動
AlphaServer 8200, 72
AlphaServer 8400, 65
AlphaServer GS160, 91
セカンダリ・コンソールの初期化
AlphaServer GS160, 91
セキュリティに関する留意事項, 43

そ
ソフト・パーティション, 30

た
ダイレクト・マップ DMA ウィンドウの変更, 175

つ
ツリー
構成, 20
ルート, 20

の
ノード
ツリーの, 20
定義, 20

は
パーティション
セットアップ, 22
ソフト, 30
ハード, 30
範囲, 21
パーティションのセットアップ, 22
ハードウェアの設定
AlphaServer 8200 への EISA 装置の取り付け, 69
AlphaServer 8400 の KFE72-DA コンソール・サブシステムの概要, 59
AlphaServer 8400 への EISA 装置の取り付け, 62
AlphaServer 8400 への KFE72-DA モジュールの取り付け, 60
ハードウェア要件
AlphaServer 4100, 75
AlphaServer 8200, 69
AlphaServer 8400, 59
AlphaServer ES40, 81
AlphaServer GS1280, 93
AlphaServer GS160, 93
GS160, 87
ハード・パーティション, 30
範囲
パーティションの, 21

ふ
ファームウェアのアップグレード
AlphaServer 4100, 77
AlphaServer 8200, 70
AlphaServer 8400, 63
AlphaServer ES40, 84
ファームウェアの入手先
GS1280, 93

め
メモリ
共通値, 189
メモリの共通値, 189

ら
ライセンス, 191

る
ルート
ツリー, 20

ろ
ローカル・エリア・ネットワーク (参照 LAN (ローカル・エリア・ネットワーク))