計算機アーキテクチャ　92−11
（１９９２．　１．　２４）

スーパーデータベースコンピュータ (SDC) における性能評価支援システム

鈴木和宏　原田昌信　平野 聡　喜連川優　高木幹雄

東京大学 生産技術研究所

概要

スーパーデータベースコンピュータ (SDC) は SQL を高速に実行する高並列リレーショナル・データベース・サーバである。本論文では SDC における、バストラフィックを解析するツールである「バスモニタ」、プロセッサ及びメモリの使用率を観察するツールである「パフォーマンスモニタ」、データを可視化するツールである「性能可視化ツール」の 3 ツールを統合した『性能評価支援システム』について、その構成及び測定結果について述べる。

The Performance Analizing Support System for The SDC, The Super Database Computer

Kazuhiro SUZUKI　Masanobu HARADA　Satoshi HIRANO
Masaru KITSUREGAWA　Mikio TAKAGI

Institute of Industrial Science, University of Tokyo

Abstract

Super Database Computer (SDC) is a high-performance relational database machine. This paper describes the Performance Analizing Support System for The SDC which is build with three tools; *Bus Monitor*, *Performance Monitor*, and *Performance Visualizer*. The first tool is used to observe the bus traffics, the percentages of processors and a shared memory are used are observed with the second tool, and the last tool is a graphical visualization tool for the above two tools. The experimental result by this system is also reported.

1 はじめに

近年の並列計算機システムの普及と共に様々なマシンアーキテクチャが設計されている。並列計算機システムの動作は複雑であり、それを正確に把握することは困難である。しかしながらシステムを効率良く動作させる為には、その性能を定量的に評価することは極めて重要なことである。並列システムの性能を評価することは今日活発に研究がなされている分野の一つであり、将来の計算機システムのハードウェアやソフトウェアの設計及び実装にとって重要な意味を持つ。

システムを評価することは、まずシステムを測定し、次に性能を解析するという二つのフェーズからなる。

第一のフェーズであるシステムの測定とは、性能を評価・解析するためのデータ（以下、性能データ）を収集することにほかならない。性能データを得る方式にはハードウェアモニタ及び、ソフトウェアモニタが考えられる [1]。ハードウェアモニタはバストラフィック等を測定するのに有効な方式であり、システムの外から測定するため、被測定システムに与える影響が小さいことが特徴である。

それに対し、ソフトウェアモニタは被測定システム内で実行環境を共有するため、オーバーヘッドを伴うが、より詳細な性能データを得ることが可能である。また、得られた性能データを「on-line」に処理し、表示する為に有効な方式である。ソフトウェアモニタにおいては、被測定プロセス中にモニタルーチンを埋め込んで性能データを測定する。そのためには、システムに用意されたライブラリを明示的に付加する方法、コンパイラに測定機能を持たせ自動的にモニタルーチンを埋め込む方法等がある [2, 3]。

第二のフェーズである、効果的なシステムの評価は、非常に複雑でしかも大量なデータが必要とされる。そのため、これらのデータを解析するためには新しい技術が必要となる。このような技術の一つが可視化である。これは得られたデータを幾何学的な図形に変換し、システム開発者にとって必要な情報を十分理解しやすい形で提供するものである。

本論文は、高並列リレーショナル・データベース・サーバである「スーパー・データベース・コンピュータ SDC」における性能評価支援システムについて述べる。本システムは「バスモニタ」、「パフォーマンスモニタ」及び、「性能可視化ツール」の 3 ツールから構成される [4, 5]。

「バスモニタ」は SDC のモジュール内の高速バスの使用率を測定するためのハードウェアモニタである。「パフォーマンスモニタ」はモジュール内の各プロセッサ及び、共有メモリの使用率を測定するためのソフトウェアモニタである。また、「性能可視化ツール」は 2 つのモニタによって得られた性能データを可視化し、性能の評価・解析に用いるソフトウエアツールである。

以上のように、本システムはハードウエアモニタとソフトウエアモニタのそれぞれの利点を生かすために、ハイブリッドモニタの構成をとる。ソフトウエアモニタにおいては、本来の処理がなされていない時にプログラム中のカウンタをインクリメントするモニタルーチンを設ける。これにより被測定系である SDC に与える影響を軽減する。

以下、2 章で SDC の概要について、ハードウェアとソフトウェアの両面から述べる。3 章で「バスモニタ」、4 章で「パフォーマンスモニタ」について、また 5 章では「性能可視化ツール」について述べる。6 章で本システムによって得られた結果を示す。

2 スーパー・データベース・コンピュータ SDC の概要

2.1 ハードウェア・アーキテクチャ

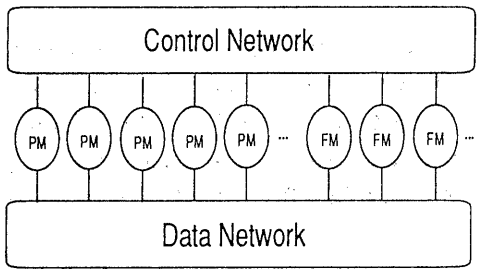

図 1: SDC の全体構成

PM データベース処理を行なうモジュール
FM フロントエンドモジュール
DNet データ転送、負荷分散を行なうデータ用の高機能オメガネットワーク
CNet 制御情報を転送するコントロール用のネットワーク

図 1 に SDC 全体の構成を示す。

スーパー・データベース・コンピュータ (SDC) は SQL を高速に実行する高並列リレーショナル・データベース・マシンである。SDC は大規模なデータベース・マシンとして、大量のディスクとプロセッサを並列に駆動し高速性を実現することを目的とする。しかしながら従来の方法では I/O ボトルネック、通信オーバーヘッド等の問題で、ディスク数、プロセッサ数の増加は必ずしも期待した特性向上をもたらさないことが明らかになった [6]。たとえば、バス共有型の密結合ではバスの飽和により、プロセッサ数を 10 台程度より増やすことは難しい。ネットワークで結合する疎結合型ではディスクからのデータ流と比較してプロセッサの処理能力が低いため、プロセッサがボトルネックとなり、同時にネットワークの通信オーバーヘッドとトラフィックの集中が問題となる。また負荷の重い結合演算などでは、演算領域を分割し、複数のプロセッサで独立に実行することが有効であるが、プロセッサ間に負荷の偏りが生ずるとシステム全体の性能が低下する。

SDC では、これらの問題を解決するため磁気ディスク装置 2 台とプロセッサ 5 台を密結合してモジュールとし、それらを高機能オメガネットワークで疎に結合する「ハイブリッド・アーキテクチャ」をとる [7, 8]。

密結合部分では軽い通信コストにより、I/O ボトルネックを解消する。また、モジュール間を負荷分散機能を有するオメガネットワークで結合することにより、モジュール数の増減によるスケーラビリティを実現する。

図 2に SDC を構成する処理モジュールの構成を示す。処理モジュールは、2 次記憶系と処理系が 2 つの共有バスを用いて密結合されている。2 つのバスのうち一方はデータを高速に転送するためのデータ転送専用バス (H-Bus) であり、他方は制御情報を転送するために用いるコントロール専用バス (C-Bus) である。これらのバスにはそれぞれ共有メモリ (D-Mem, C-Mem) が結合されており前者はステージング用に、後者はコントロール情報及び、ロック用に用いられる。

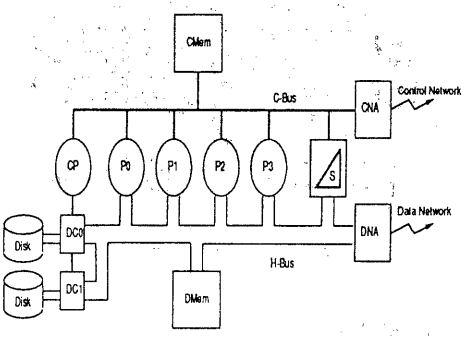

図 2: モジュールの構成

CP モジュール全体と I/O を制御するプロセッサ。
MC68020 20MHz
P0-P3 データベース処理を行なうプロセッサ群。
MC68020 20MHz
DMem ステージング用メモリ。8MByte
CMem コントロール情報、ロック用のメモリ。
DNA データ用オメガネットワーク・インターフェース
CNA コントロール・ネットワーク・インターフェース
DC0, DC1 ディスク・コントローラ
Disk0, Disk1 磁気ディスク
H-Bus データ用高速バス (40MB/sec)
C-Bus コントロール用バス

2.2 SDC のソフトウェア構成

SDC は、ひとつの関係演算の実行に当たって、並列関係演算アルゴリズム、多モジュール制御方式、プロセス・モデルの各レベルで高並列化を図る。

2.2.1 SDC のプロセス構成

図 3に SDC のプロセス構成を示す。ひとつの関係演算は Executer により、複数のプロセスによって実行される。モジュール内は、モジュール制御プロセス (PMC) によって管理され、Executer からの要求を解釈し、処理内容に従って各プロセスにコマンドを送る。データベースのデータを実際に処理するのは各データ処理プロセッサに置かれた、データ処理プロセス (DPP) である [9]。

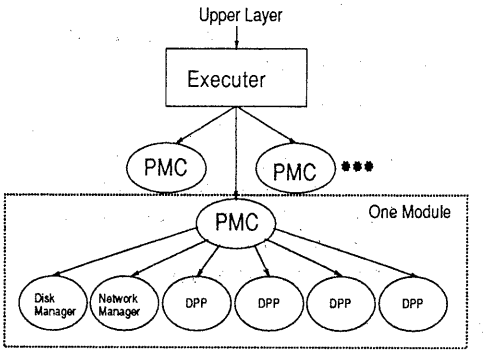

図 3: ソフトウェア構成

2.2.2 並列結合演算アルゴリズム

結合演算は、リレーションを導出する際に頻繁に用いられる演算であるが、関係演算の中でも処理負荷が特に重いため高速実行が不可欠である。結合演算のアルゴリズムは、多重ループ方式、ソートマージ方式、ハッシュ方式に分類することができる。SDC のようなマルチプロセッサマシンにおいては、この内ハッシュ結合方式が有効である。ハッシュ結合方式の特徴は、リレージョンがステージングメモリの容量を越える場合に、結合属性値をハッシュすることにより互いに結合可能性のない「バケット」と呼ぶ単位に分割し、バケット同士の結合に帰着させることである。

ここでは、GRACE Hash Join アルゴリズムについて述べる。アルゴリズムは次のような 2 つのフェーズからなる。

スプリットフェーズ

スプリットフェーズでは、ディスクに格納された入力リレーション R 及び S を B 個のバケットに分割し、中間ファイルとしてディスクに書き戻す作業を行なう。

まず、R リレーションを順次メモリ上の入力バッファに読み込み、タプルの結合属性値にスプリット関数を適用して、自モジュールの担当するバケットならば、メモリ上のバケットバッファへ格納する。他モジュールの担当するバケットの場合は、ネットワークを介して転送する。これは、4 台のプロセッサが協調して行う。他モジュールから送られてきたタプルにスプリット関数を適用してそれぞれのバケットバッファに格納する。バケットバッファがフルになると入力を中断して、ディスクに書き出す。こうして R は B 個のバケットに分割される。S リレーションについても同様にバケットに分割される。

ジョインフェーズ

R に属していたバケットから 1 つをメモリに読み込み、それに対応する S のバケットを読み込み結合を行う。これは、各モジュールが独立して行う。このフェーズは次の 2 つのフェーズをすべてのバケットについて実行することによって行われる。

ビルドフェーズ

まず、Rに属していたバケットを順次メモリ上の入力バッファに読み込みタプルの結合属性値にハッシュ関数を適用してハッシュテーブルを作成する。

プローブフェーズ

次にSのバケットを順次入力バッファに読み込み、タプルの結合属性値にハッシュ関数を適用してハッシュテーブルを探索し対応する結合属性値を発見するとタプル同士を結合して結果出力用バッファに転送する。出力バッファがフルになると、入力を中断してディスクに書き込む。

2.2.3 プロセス・モデル

SDCでは、ディスクやネットワークといったI/O処理と、プロセッサ群による処理をオーバーラップし、I/Oからのデータを実時間で処理すること、及びプロセッサ群の負荷を均等に分散することが必要である。

SDCのプロセス・モデルを図4に示す。ディスクから連続的に読み出されるデータ及び、ネットワークを転送されてきたデータは、それぞれ数個のタプルを含むタスクとして「コンテナ」と呼ぶタスク搬送用の容器に詰められる。コンテナは、仮想的なパイプを通してプロセッサに渡される。このパイプは2台のディスクとネットワークを同一とみなすことを可能にすると共に、プロセッサ群の処理能力の変動を吸収するためのバッファの役割を果たす。各バッファは複数のコンテナをリストとして管理し、空のコンテナがフリーリストを、またデータが詰められたコンテナがフルリストを構成している。

フリーリストから取得した空のコンテナにディスク及びネットワークからのデータが満たされるとフルリストの最後尾に接続され、プロセッサによる処理を待つ。データ処理プロセッサはフルリストの先端を監視し、コンテナが接続されると競合して取得して処理を行う。ここで、モジュール内のプロセッサ間での負荷分散を実現する。処理済みのデータは同様にディスクに対するタスクとしてコンテナに詰め込まれ後段のパイプへ送られる。

処理が終了し、空となったコンテナは再びフリーリストに追加される。

結合演算などにおいては処理すべきデータ量が主記憶の容量を越える場合中間リレーションを作る必要がある。この時出力バッファとして、バケットバッファと呼ぶバッファを用いる。各バケットバッファは、それぞれ固定量の空きコンテナを有するのではなく1つのフリーリストを共有しており、そこから空きコンテナを取得する。空きコンテナが次第に消費され、その数が低水位指標を下回ると空きコンテナ数の初期値の4分の3程度を回収するように各バケットを書き出す。バッファが十分な大きさに成長したバケットのみを書き出し、小量のデータしか保持していないバッファは、まとまった量になるまで書き出しを保留することで、ディスクの入出力スループットを向上させる。

ディスクの出力を行っている間に、使用されていたバッファのコンテナは順次解放されてゆくため、プロセッサ群はネットワークからのデータに対する処理を継続することが可能である。

ネットワークを介したデッドロック回避のためネットワークの出力バッファについても同様に管理している。空きコンテナがこの指標を下回ると、ディスクのリードを中断し、バケットバッファの書き出しを行う。

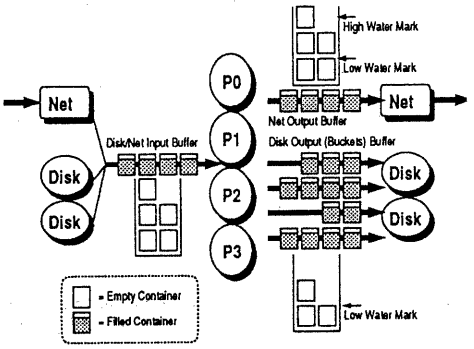

図4: プロセス・モデル

3 バスモニタ

3.1 バスモニタの構成

SDCにおける処理モジュール内には、すでに述べたようにC-Bus, H-Busの2つの共有バスが存在する。C-Busを使用する資源として、5台のプロセッサがある。バス上にはC-Busバス・アービタからそれぞれのプロセッサへ、バス使用許可信号が1本づつ計5本出ている。

一方、H-Busを使用する資源として、5台のプロセッサ、2台のディスク、およびネットワーク・インターフェースがある。バス上には、C-BusとH-Busバス・アービタからそれぞれの資源へバス使用許可信号が1本づつ計8本出ている。

バスモニタは、これら計13本のバス使用許可信号からマルチプレクサにより選ばれた4本のバス使用許可信号が、一定時間の内でアクティブになっている時間を測定し、バスの使用率を求める。

バスモニタのブロック図を図5に示す。バスモニタは、4つのカウンタ、4つのメモリ、メモリコントローラ、タイマ、マルチプレクサ及び、コントローラで構成される。

マルチプレクサは、コントローラからの指示により複数のバス使用許可信号の内、測定の対象となる4つを選び出す。この時、マルチプレクサは、『全ての処理用プロセッサのH-Bus使用率』のように、複数のバス使用許可信号を加算する機能も備えている。

カウンタはマルチプレクサによって選択された信号が、アクティブになっている時間を計測する。カウンタのビット数

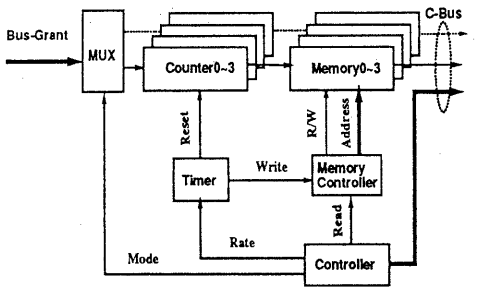

図 5: バスモニタのブロック図

は8ビットである。1ビット当たりの時間分解能は、コントローラからの指示により50nsもしくは100nsが選択される。これは、SDCにおけるシステムクロックが50 nsであり、C-Busは非同期バスを、H-Busはバスサイクル100nsの同期バスを採用しているためであり、どちらのバスを詳しく測定するかによって、1ビット当たりの時間分解能を変更可能となっている。

メモリはカウンタの計測値を記録する。容量はそれぞれ2 Mbyteである。

メモリコントローラは、タイマからの書き込み要求や、コントローラからの読み出し要求により、メモリに与えるアドレスの管理、およびリード、ライトの制御を行なう。

タイマは一定時間ごとにメモリコントローラに対し書き込み要求を出すと同時に、カウンタをリセットする。

最後にコントローラは、バスモニタ全体の制御、およびC-Busとのインターフェースを司る。

3.2 測定時間

次に、バスモニタの測定可能時間について述べる。バスモニタはタイマがフルカウントになるたびに、カウンタの内容をメモリへ書き込む。よって最大測定可能時間は、すべてのメモリに測定結果を書き込む時間となる。

タイマのビット数は、8、9、10ビットの内一つが選択される。また、1ビット当たりの時間分解能は、カウンタと同一で50nsもしくは100nsである。

いま、タイマを8ビットとし、1ビット当たりの時間分解能を100 nsにした時、カウンタの値がメモリに書き込まれる時間間隔は、

$$100\ \text{ns} \times 2^8 \simeq 25.6\mu s$$

よって、測定可能時間は、メモリの容量が2Mbyteであるため、

$$25.6\mu s \times 2\text{Mbyte} \simeq 54\ \text{秒}$$

となる。

またこの時、測定され得るバス使用率の上限は、カウンタのビット数もタイマと同じく8ビットであるため100%となる。

また、タイマのビット数が9ビット、10ビットと増加するに従い、メモリにカウンタの値が書き込まれる間隔が長くなるため、測定可能時間は約108、216秒と長くなる。それと同時に、測定され得るバス使用率の上限は50、25 %と減少する。

一方、1ビット当たりの時間分解能を50 nsにした時の、測定可能時間は、タイマのビット数により約27、54、108秒となる。

以上を表1にまとめる。

ビット数	時間分解能		フルスケール
	50ns	100ns	
8	27秒	54秒	100%
9	54秒	108秒	50%
10	108秒	216秒	25%

表 1: バスモニタの測定時間

4 パフォーマンスモニタの機能と構成

4.1 性能データの収集

パフォーマンスモニタはSDC上のプロセスの中に各種のモニタルーチンを備えて、性能データの収集を行う。本ツールは、プロセッサ及びステージングメモリの使用率を測定する。以下にそれぞれの性能データの測定方式について述べる。

- プロセッサの使用率
  SDCのプロセッサ群は、共有メモリ上に用意された仮想的なパイプの出口に到着したタスクを競合して取得し、そのタスクの処理を行う。パイプの出口にタスクが到着していない時はタスクの到着を待つ。そこで、タスクの到着待ちをする際にプログラム内に用意したカウンタをインクリメントするようなルーチンを付加する。
  したがってこのカウンタは、プロセッサのアイドル状態を調べるための指標となる。たとえば、プロセッサの処理能力よりもデータ流が十分大きい時は、プロセッサは常時タスクを取得することができるためにカウンタの値は一度もインクリメントされることなく0を示す。逆にタスクが全く到着しない場合、カウンタの値は最大値となり、プロセッサの使用量を知ることが可能となる。
  このことから、プロセッサの使用率はこれらのカウンタの値を一定時間ごとに読み出し、最大値と比較することによってを求めることができる。
- ステージングメモリの使用率
  データメモリは、入力用、結果出力用、中間ファイル出力用、及びネットワーク出力用のそれぞれのバッファとして用いられる。各バッファは図6に示すように複数のコンテナをリスト構造によって管理している。コンテナのヘッダ部分はコントロールメモリ上に格納され、データメモリ上にはデータ部分のみが存在する。
  このコンテナに関する管理情報、すなわちフルリストに接続されたバッファの数をコントロールメモリから調べることによってデータメモリの各バッファの使用率を算

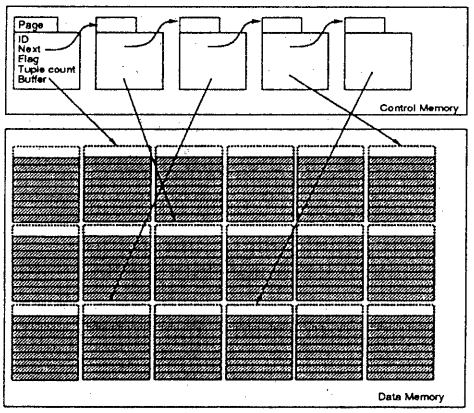

図 6: コンテナの管理

出する。

4.2 構成

パフォーマンスモニタは、図7に示すようにSDCのCp上で動作するSampler及びエンジニアリングワークステーションSUN上で動作するCollectorによって構成される。

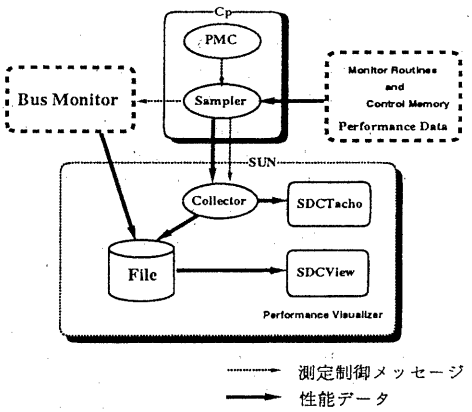

図 7: パフォーマンスモニタの構成

Samplerは、PMCから送られる測定開始要求メッセージを受け性能データの測定を開始する。測定は一定時間ごとに性能データを読み出し、それらをネットワークを通してCollectorに送出することにより行われる。読み出された性能データをその都度転送すると、プロセッサ及びネットワークの負荷が増大し、システムに与える影響が大きくなることが懸念される。そこで本ツールでは10msecごとに性能データを読み出し、それを30回収集するごとに送出することでシステムに与える影響を軽減する。

またSamplerはバスモニタの制御も担当し、SDCの動作と同期してバスモニタによる測定を開始することが可能である。

Samplerは PMCからの測定終了要求メッセージを受けた時に測定を終了し、Collectorに対し性能データ収集の終了を知らせる。この時バスモニタの終了を待ってバスモニタ上の測定データの前処理を行い、その結果をネットワークを通してSUNへ転送する。

Collectorは SUN上で Samplerから転送される測定データを受けとり、SDCViewが必要とするフォーマットに変換した後、性能データをファイルとして格納すると共にSDCTachoに対して「on-line」データを供給する。

5 SDCにおける性能可視化ツール

SDCにおける性能可視化ツールは収集された性能データを静的に解析するための可視化ツールSDCView及び、収集と同時に性能データを観測するためのSDCTachoによって構成される。以下にそれぞれのツールについて述べる。

5.1 SDCView

バスモニタ及び、パフォーマンスモニタは専用の可視化ツールであるSDCViewによって性能データを可視化する。SDCViewは性能データが格納されたファイルを読み込むと、図8に示されるようなウィンドウを表示する「off-line」ツールである。

ウィンドウ内のグラフは横軸に秒を単位とした時間を、また縦軸にはそれぞれの資源の使用率を百分率で示す。各性能データは縦に並べて表示され、処理の進行にそった定量的な解析を容易にする。ウィンドウシステムとしては、X window systemを用いる。

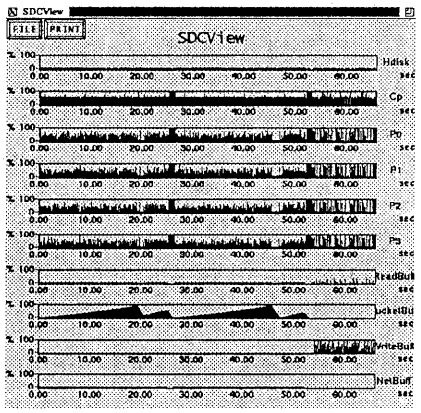

図 8: SDCViewの概観

SDCViewは次のユーザインターフェースを有する。

- グラフの拡大

  ウィンドウ内でマウスを、ボタン1を押したまま移動し

て放す(ドラッギング)ことによって、新たなウィンドウにグラフを拡大・表示することができる。

- 時間の表示
  同様にボタン2によって二点間の経過時間がポップアップウィンドウに表示される。
- 入力ファイルの選択
  ウィンドウ内のボタンを操作することによって、所望のファイルを選択する。
- ハードコピー
  ページ記述言語 PostScript によりプリンタデバイスに出力する。
  またドローイングエディタ idraw の形式で出力することも可能である。

5.2 SDCTacho

パフォーマンスモニタによって得られる性能データを動的に可視化するツールがSDCTacho である。SDCTacho は図9に示されるようなウィンドウ内に、Collector から送られるon-line データをリアルタイムに表示する。すなわち性能データをSDC の実行と同時に観察することを可能とするツールである。各性能データは最大値を 100% とするアナログメーターとして表示され、その針の振れによって性能データの変化が表現される。

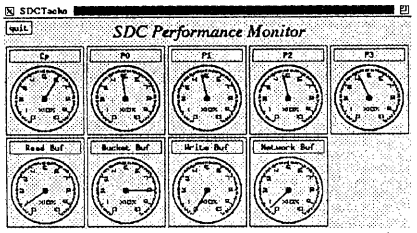

図 9: SDCTacho の概観

6 測定結果

6.1 バスモニタによる高速バスの使用率の測定

図10にバスグラフによって得られた性能データを SDC-View を用いて可視化した例を示す。グラフは拡張ウィンスコンシン・ベンチマーク・テストを2モジュールで行った時に得られた性能データを示したものである。ベンチマーク・テストの条件は、タプル長208バイト、タプル数100万件、選択率10% の結合演算である。

グラフは、H-Bus の使用率を示しており、各データは上からディスク、処理プロセッサ群、及びネットワークである。R、S で示される部分が各リレーションのスプリットフェーズであり、Join で示された部分がジョインフェーズを示している。

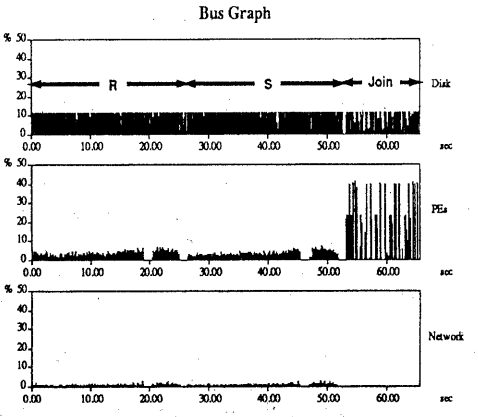

図 10: H-Bus の使用率

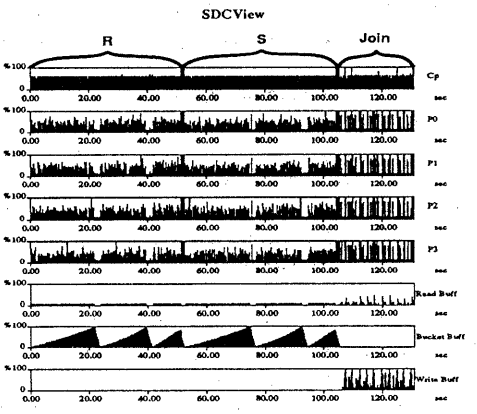

図 11: 各資源の使用率

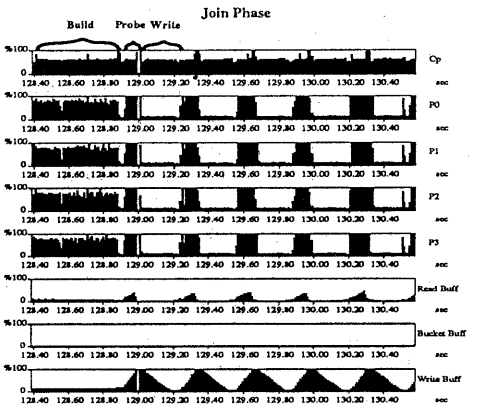

図 12: ジョインフェーズの拡大図

この結果から、H-Bus は使用率がもっとも高いジョインフェーズにおいても、今だ余力を残しており、バスの飽和による性能低下は起きていないことが確認された。

6.2 パフォーマンスモニタによる各資源の使用率の測定

図 11にパフォーマンスモニタによって得られた性能データを、同様に SDCView によって可視化した例を示す。グラフは、上記のベンチマーク・テストを 1 モジュールで行った結果である。各データは上から、Cp、P0 ～ P3、リードバッファ、バケットバッファ、ライトバッファの使用率である。

スプリットフェーズ (S, R) では P0 ～ P3 のグラフは、ほとんど同じ形をしておりモジュール内での負荷分散が十分実現されていることがわかる。したがって、図 4に示されたプロセスモデルによる並列化が有効に動作していることが確認された。また使用率は 60% 程度の値を示しており、プロセッサ群はまだ余力を残していることがわかる。リードバッファは、ほぼ一定の値で約 10% の使用率である。このことからも、プロセッサ群による処理が I/O の速度に追随していることがわかる。

バケットバッファの使用率は直線的に増加し、100% になると、減少している。これによって、読み込んだリレーションをバケットに分割する際、バケットバッファがフルとなったため、読み込みが中断され各バケットが中間ファイルとして書き出されているということが直観的に理解できる。また、バッファフルの状態が続くことなく空きコンテナが回復していることから、低水位指標が有効に作用していることが確認された。

Join で示されたジョインフェーズを拡大したものが図 12である。グラフは 1 組のバケットの結合演算部分を示しており、Build で示された部分がビルドフェーズに、Probe で示された部分がプローブフェーズに対応している。また Write で示される部分が結果出力部である。

ビルドフェーズでは、プロセッサの使用率はスプリットフェーズよりも高い値を示しているが 100% には到っていない。それはリードバッファの使用率が、スプリットフェーズとほとんど変化がないことからもわかる。

プローブフェーズでは、プロセッサは 100% の使用率を示しており、リードバッファが約 40% まで使用されている。これは、プロセッサの処理が I/O に間に合わず、ディスクから読み込まれたデータがバッファリングされるためである。

6.3 パフォーマンスモニタによる影響

パフォーマンスモニタは純粋なソフトウェアモニタであるため、SDC に対する影響が懸念される。そこで、パフォーマンスモニタを使用した場合と使用しない場合のベンチマーク・テストの実行時間を次に示す。

	使用時	未使用時
実行時間 (sec)	130.92	130.50

これにより、パフォーマンスモニタの影響は約 0.3% であり、懸念されたパフォーマンスの低下は極めて小さいものであることが確認された。

7 おわりに

本論文では、スーパーデータベースコンピュータ SDC における性能評価支援システムを製作し、処理モジュール内のバストラッフィック、プロセッサ及びステージングメモリの使用率を測定した結果について述べた。

バスモニタによって、モジュール内の共有バスはいまだ余力を残しており、バスの飽和による性能低下はおきていないことが確認された。またパフォーマンスモニタによる観測から、モジュール内での負荷分散が実現されていることが確認された。最後にパフォーマンスモニタが SDC に与える影響は極めて小さく、本ツールの有用性が確認された。

今回は GRACE Hash Join アルゴリズムを用いた結合演算を行った時の測定に留まったので、今後は他の関係演算についてさらに詳しく性能評価を進める予定である。

参考文献

[1] D. Wybranietz and D. Haban. Monitoring and Measuring Distributed Systems. *Parallel Computer Systems: Performance Instrumentation and Visualization, Margaret Simmons, Rebecca Koskela, eds. ACM*, 1990.

[2] A. D. Malony. JED: Just an Event Display. *Parallel Computer Systems: Performance Instrumentation and Visualization, Margaret Simmons, Rebecca Koskela, eds. ACM*, 1990.

[3] K. B. Kenny and K. Lin. Measuring and Analyzing Real-Time Performance. *IEEE SoftWare*, sep., pp. 41–49, 1991.

[4] 原田, 鈴木, 平野, 喜連川, 高木. スーパーデータベースコンピュータ (SDC) のバスモニタ. 情報処理学会第 *42* 回全国大会, 1991.

[5] 鈴木, 平野, 喜連川, 高木. スーパーデータベースコンピュータ (SDC) における性能評価ツール. 情報処理学会第 *43* 回全国大会, 1991.

[6] M. Kitsuregawa, M. Nakano, L. Harada, and M. Takagi. Functional disk system for relational database. *Proc. of 3d Int. Conf. on Data Engineering*, pp. 88–95, 1987.

[7] 平野, 原田, 中村, 小川, 楊, 喜連川, 高木. スーパーデータベースコンピュータ SDC のアーキテクチャ. 並列処理シンポジウム *JSPP*, pp. 137, 1990.

[8] 喜連川, 小川. バケット平坦化機能を有するオメガネットワーク. 情報処理学会論文誌, Vol. 30, No. 11, pp. 1494, 1989.

[9] 平野, 楊, 喜連川, 高木. スーパーデータベースコンピュータ SDC のシステムソフトウエアの概要. 情報処理学会第 *39* 回全国大会, 1989.