ハイパフォーマンス
コンピューティング 67－14
（1997. 8. 19）

ギャングスケジューリングの PC クラスタ上での実装

堀 敦史† 手塚宏史† 石川 裕†

我々は，メッセージ通信型のプログラミングを対象としたギャングスケジューラを開発した．本稿は，開発したギャングスケジューラのスケジューリングオーバヘッドについて，PC クラスタ上で評価した結果について報告するものである．評価の目的は，アプリケーションの通信特性の違いがギャングスケジューリングのオーバヘッドに与える影響，実装された方式のスケーラビリティ，および co-scheduling skew の 3 点に注目した．今回の評価の結果，通信の量による違いよりも，co-scheduling skew の影響が大きいことが判明した．スケーラビリティの面においては，2 プロセッサで動作する並列アプリケーションで比較的大きなオーバヘッドが見られたが，多くの場合，100 *msec* の量子時間でオーバヘッドは 10 % 以下であり，4 台以上の場合では，スケーラビリティに問題は見られなかった．

Gang Scheduling Implementation on PC Cluster

Atsushi HORI ,† Hiroshi TEZUKA † and Yutaka ISHIKAWA†

We have developed a gang-scheduler for message passing programs. This paper reports on evaluation results of our gang-scheduler running on a PC cluster. We focus on, i) how communication patterns affects gang scheduling overhead, ii) scalability, and iii) co-scheduling skew. Through the evaluation, we found that effect of co-scheduling skew can be larger than effect of the amount of messages communicating. Also we found larger overhead with applications running on two processors, however, we could not find any scalability problem with applications running more than two processors. In most cases, implemented gang scheduling overhead is less than 10 % with 100 *msec* time quantum.

1. はじめに

我々は SparcStation20 を Myrinet で接続したワークステーションクラスタ上に SCore-D と呼ばれるギャングスケジューラを構築し，その結果について既に報告している[1),2)]．しかしながら SunOS 固有のシグナル配送の遅れが原因で，ギャングスケジューリングそのものの性能を計測することができなかった[1)]．我々は別途 PC クラスタを開発し[3),4)]，SCore-D を PC クラスタに移植した。本稿は PC クラスタ上での計測した SCore-D ギャングスケジューリングの性能についての結果について報告するものである．

ギャングスケジューラとしての SCore-D を設計開発するにあたり，我々は以下の点に十分留意した．

ユーザプログラムの性能重視

ギャングスケジューラを実現するために，ユーザプログラム（ギャングスケジューリングの対象となるプログラム）の実行効率低下を可能な限り避けることが重要と考える．このために通信ハードウェアをユーザプログラムから直接制御することを許すユーザレベル通信とし，ユーザプログラムにおける通信性能を最大限に引き出すと同時に，ギャングスケジューリング時に通信ハードウェアの状態を退避復帰している．

ギャングスケジューリングを実装するためにユーザプログラム通常の実行性能が損なわれるようなことは出来る限り避けるべきと考える．ギャングスケジューリングは時分割スケジューリングを可能にする．システムに単一のジョブしか与えられなかった場合にはギャングスケジューリングは不要である．ギャングスケジューリングに起因するオーバヘッドは全てギャングスケジューリング時に発生するようにすれば，システムの負荷が軽い時にはバッチスケジューリングと同等の実行効率が得られることになる．

ギャングスケジューリングオーバヘッドの低減

我々は，ワークステーションクラスタや PC クラスタの構築を通じ，従来の MPP に匹敵する性能の並列計算機が近い将来より安価になるとの想定し，その結果，並列対話プログラミング環境の必要性が生じると考えている．ただし，対話の粒度は逐次の場合よりも大きいと考え，ギャングスケジュー

† 新情報処理開発機構つくば研究センタ
Tsukuba Research Center, Real World Computing Partnership

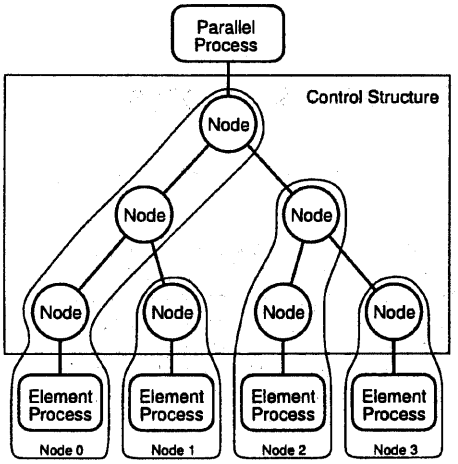

図1 並列プロセスの制御構造

リングの時分割量子時間は 1 秒程度を想定している．したがってギャングスケジューリングのオーバヘッドは 1 秒程度の量子時間で問題にならない程度に十分小さいことを目標としている．

今回の計測の目的は，我々が実装したギャングスケジューラの *i)* オーバヘッドの絶対値，*ii)* スケーラビリティ，*iii)* オーバヘッドの主要因，および，*iv)* アプリケーションの通信特性がオーバヘッドに与える影響，の 3 点である．

2. 実　　装

SCore-D は，クラスタを構成する個々のワークステーションや PC 上の UNIX の下で走るデーモンプロセスの集合であり，クラスタ全体を統合し，単一なシステムイメージをユーザに提供するユーザレベル並列オペレーティングシステムである．ワークステーションや PC を結合する通信ハードウェアとしては Myricom 社の Myrinet が想定されており，Myrinet のホストインターフェイスには我々が開発した PM[5),6)] と呼ばれる低レベル通信ソフトウェアを用いる．

PM はギャングスケジューリングの実装を考慮して設計されている．具体的には，*i)* 「チャネル」と呼ばれるネットワークのソフトウェア的な多重化，*ii)* チャネルの状態のメモリへの退避およびメモリからの復帰，*iii)* あるノードから送信された全てのメッセージがネットワーク中に存在しないことの確認，といった機能がサポートされている．PM はソフトウェアによる流量制御を行なっている．ネットワーク中に送信されたメッセージの存在確認は，流量制御のための Ack/Nack の返答メッセージの応用して実現されている．

ユーザプログラムは，SCore-D の子プロセスとして実行され，UNIX のシグナルによりその実行が制御される．同じユーザプログラムから生成されたプロセス群は「並列プロセス」と呼ばれ，SCore-D が全体を並列プロセス全体を管理する．並列プロセスと個々のプロセスはツリー状の分散構造で管理されている（図 1）．この制御構造のルートには「並列プロセス構造体」があり，制御構造のリーフの部分は「要素プロセス構造体」がある．これらの構造体は，並列プロセスや要素プロセスの状態や管理情報を保持している．制御構造は，ギャングスケジューリングのための並列プロセス全体の制御や，個々のプロセスで発生する終了やシグナルなどの事象を効率的に管理することを目的としている．制御構造ツリーの各ノードは状態を持つ．プロセスの状態変化は全てのプロセスの停止など同期が必要な場合と，終了や例外シグナルなどの非同期的なイベントがある．前者の場合は，制御構造のノードがサブノードの同期を取る．後者の場合は，ノードがスーパーノードに通知すると同時にノードに非同期イベントを記憶しておき，同じ種類のイベントが別のサブノードで発生しても，スーパーノードに同じ種類のイベントの発生を 2 回以上通知しない．これはイベント通知がルートに集中するのを避けるためである．

SCore-D は C++ にマルチスレッド拡張を施した MPC++[7),8)] で記述されている．MPC++ ランタイムライブラリ[9)] は PM を用いており，リモート関数（スレッド）起動，グローバルポインタ，リモートメモリの読み書き，同期構造体などの並列プリミティブが提供されている．上記の分散制御は，グローバルポインタを用いて実現されている．

ギャングスケジューラは，この制御構造を用いて 4 つのフェーズから成る手順に従って並列プロセスを制御する（図 2）．

(1) Freeze フェーズ全てのプロセスに `SIGSTOP` を送り，プロセスを停止させる．`wait()` によりプロセスの停止を確認した後，ユーザプロセスが用いている PM チャネルから送信されたメッセージがネットワークからなくなるまで待つ．

(2) Save フェーズ各プロセスが用いている PM チャネルの状態をメモリに退避する．退避完了の同期により，それまでに走行していた並列プロセスは停止したとみなされる．

(3) Restore フェーズ新たにスケジューリングする並列プロセスが用いていた PM チャネルの状態をメモリから復帰する．

(4) Run フェーズ全ての PM チャネルの状態が復帰された後，個々のプロセスに対し `SIGCONT` を送る．この結果，並列プロセスは実行状態になる．

Run フェーズを除く 3 つのフェーズの終了は全て制御構造において同期される．Freeze フェーズにおいて，最終的に並列プロセス全体が停止し，その並列プロセスから発せられたメッセージがネットワーク中に存在しないことが確認される．次の save フェーズにおい

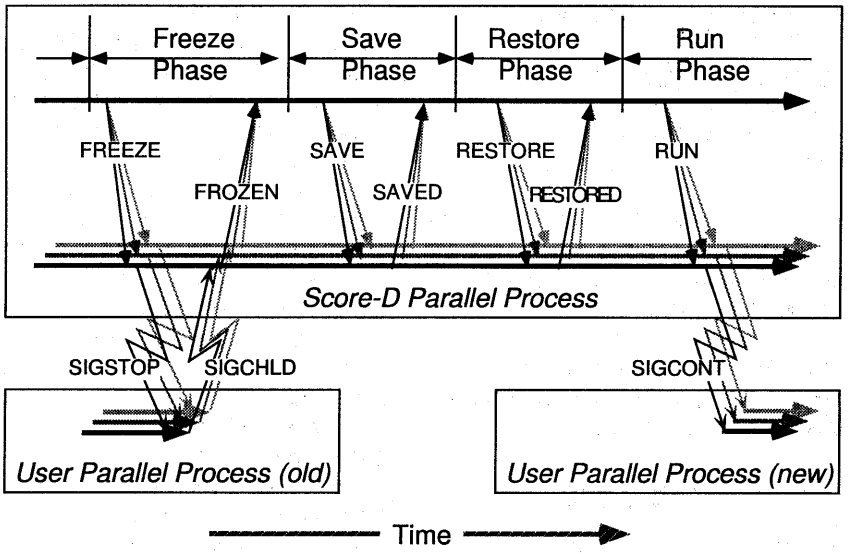

図2 ギャングスケジューリング手順

て，停止した並列プロセスのメッセージが存在しないということは，その並列プロセスに属するいかなるプロセスも，これ以降メッセージを受信しないということであり，各プロセスにおける PM チャネルの状態が確定したことを意味する．従って，PM チャネルの状態を安全に退避することができる．

PM のチャネルの状態（コンテキスト）は送受信のための FIFO バッファが大半を占めている．このため，save フェーズと restore フェーズの処理時間は，そのチャネルの FIFO バッファに含まれている通信メッセージの数と総量に依存する．また，freeze フェーズの時間はその時点でネットワーク中に存在するメッセージ量に依存する．このことから，SCore-D のギャングスケジューリングのオーバヘッドはアプリケーションの通信特性に大きく依存すること考えられる．

表1 PC クラスタの仕様

Node Processor	Pentium
Clock [MHz]	166
Memory [MB]	64
Number of Nodes	32
Link Speed [MB/s]	160

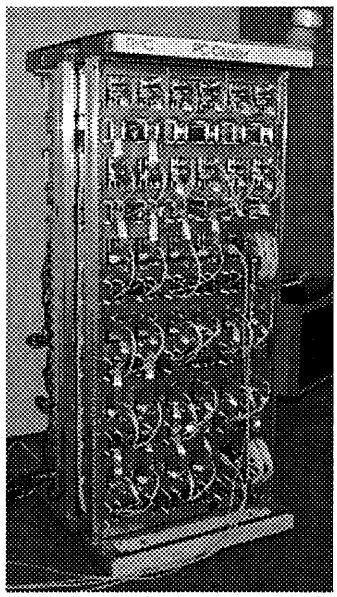

図3 PC クラスタ

3. 評　　価

評価に用いた PC クラスタの外観を図 3 に，そのハードウェアの主要諸元を表 1に示す．以下，本稿における計測は全てこの PC クラスタを用いた．また，計測に先立ち，最も影響の大きいと思われる update デーモンは終了させてある．

3.1 基本性能

実際のギャングスケジューリングの評価の前に，PM および SCore-D の基本性能を計測した．

表 2 は，PM 単体でのチャネルコンテキストの退避と復帰に要する時間である．PM のバッファの大きさは，送信バッファには最大 512 メッセージ，メッセージ総量の最大 48 KB，受信バッファには最大 4,096

表2 チャネルコンテキストの退避／復帰時間

Buffer Status	Save [msec]	Restore [msec]
Both Empty	0.50	0.17
Send Full	2.75	2.12
Recv Full	2.76	2.30
Both Full	5.08	4.32

メッセージ，メッセージ総量の最大 64 KB となっている．表中，"Both Empty" とあるのは送受信双方のバッファとも空，"Send Full" とあるのは送信バッファが満杯の状態，"Recv Full" とあるのは受信バッファが満杯，"Both Full" とあるのは両方のバッファとも満杯の状態を意味する．退避の時間の方が長いのは，プロセッサから見た I/O デバイスからの読み込み速度が書き速度よりも遅いためである．

表3 制御のための通信時間 [μsec]

# of Branches	Number of Nodes 2	4	8	16	32
2	74.0	195.4	278.5	434.4	509.9
4	73.3	106.9	231.3	**248.7**	357.0
8	**72.5**	**106.0**	145.3	270.3	**289.1**
16	72.8	106.5	145.6	258.2	362.4
32	73.1	124.7	**144.7**	257.9	494.3

表 3 は，実際に並列プロセスの制御に用いている制御構造に沿った通信時間を，プロセッサ台数と分散木構造の枝の数を変えて調べたものである．ここでは具体的な処理は何もせずに，ルートから全てのリーフへの伝達と，その返りを同期で待つ，という処理に要する時間だけを計測した．この結果から，以下のギャングスケジューリングの全ての計測において，制御構造は 8 分木とした．実際のスケジューリング処理においては，ユーザプロセスと並行して走るため，この表に示した値よりもプロセス切替の時間が余分にかかるものと思われる．

3.2 ギャングスケジューリングのオーバヘッド

ギャングスケジューリングのオーバヘッドを計測するために評価プログラムを用意した．全く通信せずに各プロセッサ上で独立した計算を繰り返すもの (EP)，ひたすらバリア同期を繰り返すもの (BAR)，そして，全てのプロセッサが 1 MB のデータを 2 KB のパケット，512 個に分割してブロードキャストしてからバリア同期をとることを繰り返すもの (BC) の 3 つである．このブロードキャストは PM が提供する機能で，一回のメッセージコピーにより複数に相手に送ることができる．それぞれ計測アプリケーションは，実アプリケーションの局面を誇張したものであるため，実アプリケーションで計測するよりもギャングスケジューリングのオーバヘッドと通信特性の関係が明確になると期待される．

各フェーズの処理時間

図 4 は，SCore-D の内部処理，freeze, save そして restore の時間をそれぞれの計測プログラムにおいて，プロセッサ台数を変化させて計測した結果である．グラフの縦軸は処理時間であり，3 つのグラフは全て同じスケールになっている．

計測プログラムにおいて，EP < BAR < BC の順で通信メッセージの量が多いと考えられる．プロセッサ台数が 8 以上の場合、この関係はそのままグラフに現れている．しかしながら，プロセッサ台数が 4 以下における BAR と BC 計測プログラムの計測結果では，必ずしも推定されるメッセージ量とスケジューリング処理の時間関係が対応していない．現在のところこの理由については不明である．

表 2 の値と図 4 の値を較べると，メッセージ量が最も多いと思われる 32 プロセッサにおける BC 計測プログラムでさえも PM のバッファの半分以下しか利用していないものと推測される．

並列プロセス切替の処理時間の 2/3 以上はチャネルコンテキストの退避と復帰の時間に費やされている．これは，PM が送受信合わせて 100 KB 以上の大きなメッセージバッファを持っているため，メッセージバッファの内容の退避復帰に時間がかかること，PCI バス上にある Myrinet インターフェイスの内部状態のアクセスが遅いことが原因と考えられる．

アプリケーションから見たオーバヘッド

アプリケーションから見たギャングスケジューリングのオーバヘッド (O) は，ギャングスケジューリングなしの場合の計測プログラムの実行時間 (T_{NoGang}) とギャングスケジューリング下での実行時間 (T_{Gang}) から次式により求めた．

$$O = (T_{Gang} - T_{NoGang})/T_{NoGang}$$

ここで計算された値は T_{NoGang} と T_{Gang} の時間差が少ない場合，計測誤差の影響を受け易いことに注意を要する．図 5 は，この様にして計測された結果である．時分割量子時間は 100 $msec$ と 200 $msec$ の 2 つのケースで計測した．この図においても 3 つのグラフの縦軸のスケールは同じである．

EP プログラムでは SCore-D の内部処理時間に見合ったオーバヘッドが計測されている．また，量子時間が倍になると，オーバヘッドもほぼ半分になることが確認された．内部処理時間の合計よりも観測されたオーバヘッドが大きいのは，SCore-D と計測プログラムの間で生じるプロセス切替，キャッシュ効果，などが考えられる．

BAR 計測プログラムでは，これらに加え co-scheduling skew[10] が加わると推測される．BAR プログラムで観測されたオーバヘッドは，SCore-D の内部処理時間の合計よりも大きい．この大半は co-scheduling skew により生じたものと考えられる．

BC アプリケーションではネットワークに大きな負荷

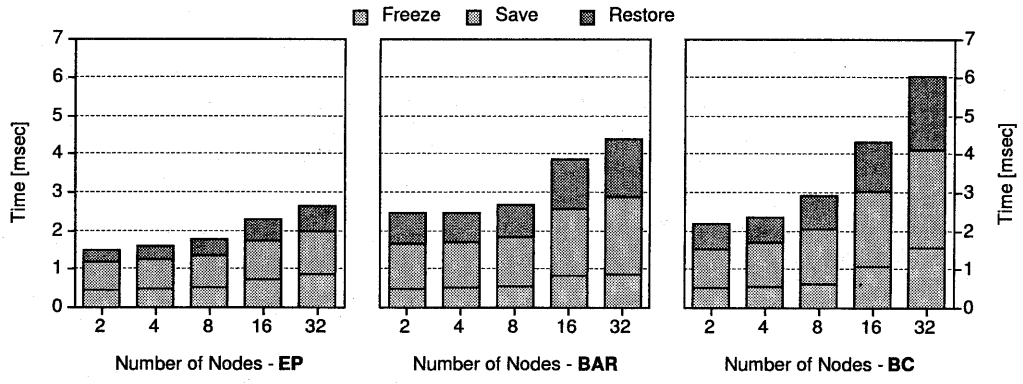

図4 Freeze, save および restore フェーズの処理時間

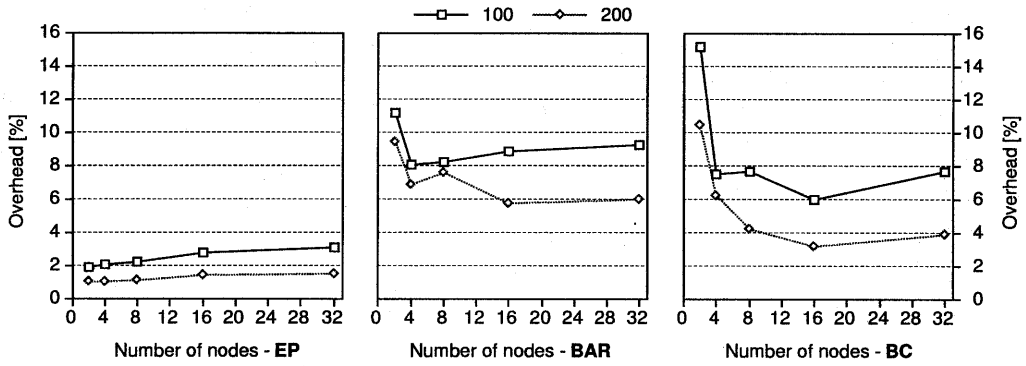

図5 アプリケーションから見たスケジューリングオーバヘッド

をかける．このためネットワーク中にメッセージが滞り，挙動が不安定になりがちになる．プロセッサ台数が 2 の場合には，オーバヘッドは 10 % を越える大きな値になっている．少ないプロセッサ台数の時に，互いにメッセージをやりとりするような場合，互いの送受信が同期し合うことがある．SCore-D によりギャングスケジューリングの結果，co-scheduling skew などにより同期に乱れが生じ，結果として遅くなったものと推測される．BAR 計測プログラムにおけるプロセッサ台数が 2 の場合も同じ理由と考えられる．これは，ギャングスケジューリングを行なわない場合でも，余分な処理を加えることで良く似た結果が生じることを確認している．

しかしながら，プロセッサ台数が 16 以上の場合に限れば，スケーラビリティに関する問題は見られない．BAR 計測プログラムで顕著に現れていると思われる co-scheduling skew については，プロセッサ台数が増えるにつれその影響は頭打ちになると考えられる．BC においては通信量がプロセッサ数の 2 乗のオーダーで増えるので，その分ギャングスケジューリングのオーバヘッド増えるものと推測される．図 5 からは，プロセッサ台数が 32 以下の場合，メッセージの量よりも co-scheduling skew の影響の方が大きいと考えられる．

4. 関連研究

Thinking Machines 社の CM-5 には "All-Fall-Down" と呼ばれるハードウェア機構によりギャングスケジューリングが支援されている．しかしながら，All-Fall-Down 機構にも関わらず，CM-5 のギャングスケジューリングのオーバヘッドについては "CM-5 incurrs a minimum overhead of 4*ms*, with typical times closer 10*ms*." という報告がある[11]．この 4 ～ 10 *msec* という値は，今回計測された SCore-D のオーバヘッドと同等である．Myrinet にはギャングスケジューリングを支援する機構は何もない．SCore-D および PM によるギャングスケジューリングの実装では，全てソフトウェアで実現されている．このため CM-5 では不可能であったパーティションの動的な変更が可能となっており，より柔軟なジョブスケジューリングが可能になっている*．

* このようなジョブスケジューリングの例として DQT[12] があ

Franke らは，IBM 社の SP-2 上に SHARE と呼ばれるギャングスケジューラを実装している[13]．SHARE では並列プロセス切替後に到着した直前の並列プロセスのメッセージは捨てる方式である．捨てられたメッセージは reliable プロトコルにより回復される．これに対し，SCore-D は並列プロセスのコンテキスト切替時に積極的にネットワーク中のメッセージを flush している．これは Myrinet の信頼性が十分高いと仮定し，信頼性回復のためのプロトコルを実装しない分，より高い通信性能を実現しようとしているからである．

ま と め

我々は SCore-D と呼ばれるギャングスケジューラを開発し，それを PC クラスタ上で評価した．SCore-D は，メッセージ通信型のプログラムを対象といしているため，プロセス毎にメッセージバッファの管理が必要である．我々は通信における流量制御プロトコルを応用し，ネットワーク中にギャングスケジューリングの対象となる並列プロセスのメッセージが存在しないことを確認し，メッセージバッファの内容を退避復帰させることでギャングスケジューリングを実現した．

評価の結果，ギャングスケジューリングのオーバヘッドはユーザプログラムの通信パターンの影響を強く受けることが判明した．これはギャングスケジューリングの実装方式から推測された事である．しかしながら，co-scheduling skew は間接的な観測の結果，無視できない程大きいと思われる．SCore-D の設計においてスケーラビリティの確保は優先度の高い目標であったが，今回の計測からは満足できる結果が得られたと考える．今後は，より大規模な PC クラスタを構築し，スケーラビリティに十点を置いた評価を続けると同時に，実アプリケーションを用いた評価を行なう予定である．

参 考 文 献

1) 堀敦史, 手塚宏史, 石川裕, 曽田哲之, 小中裕喜, 前田宗則: 並列プログラム実行環境のワークステーションクラスタ上での実装, 並列処理シンポジウム JSPP'96, 情報処理学会 (1996).
2) Hori, A., Tezuka, H., Ishikawa, Y., Soda, N., Konaka, H. and Maeda, M.: Implementation of Gang-Scheduling on Workstation Cluster, *IPPS'96 Workshop on Job Scheduling Strategies for Parallel Processing* (Feitelson, D. G. and Rudolph, L.(eds.)), Lecture Notes in Computer Science, Vol. 1162, Springer-Verlag, pp. 76–83 (1996).
3) 手塚宏史, 堀敦史, 石川裕, 曽田哲之, 原田浩, 古田敦, 山田努: PC とギガビット LAN による PC クラスタの構築, 計算機アーキテクチャ研究会資料, 96-ARC-119, 情報処理学会, pp. 37–42 (1996).
4) Hori, A. and Tezuka, H.: Hardware Design and Implementation of PC Cluster, Technical Report TR–96017, RWC (1996).
5) 手塚宏史, 堀敦史, 石川裕: ワークステーションクラスタ用通信ライブラリ PM の設計と実装, 並列処理シンポジウム JSPP'96, 情報処理学会, pp. 41–48 (1996).
6) Tezuka, H., Hori, A., Ishikawa, Y. and Sato, M.: PM: A Operating System Coordinated High Performance Communication Library, *High-Performance Computing and Networking '97* (1997).
7) 石川裕, 堀敦史, 手塚宏史, 佐藤三久, 松田元彦, 小中裕喜, 前田宗則, 友清孝志: 並列プログラミング言語 MPC++ のワークステーションクラスタ上での実現, コンピュータシステムシンポジウム, pp. 33–38 (1995).
8) Ishikawa, Y., Hori, A., Tezuka, H., Matsuda, M., Konaka, H., Maeda, M., Tomokiyo, T. and Nolte, J.: MPC++, *Parallel Programming Using C++* (Wilson, G.V. and Lu, P.(eds.)), MIT Press, pp. 429–464 (1996).
9) 堀敦史, 手塚宏史, 石川裕, 高橋俊行, 曽田哲之, 堀川勉, 小中裕喜, 前田宗則: マルチスレッド言語のための実行時ライブラリの実装, 計算機アーキテクチャ研究会資料, 96-ARC-117, 情報処理学会, pp. 37–42 (1996).
10) Arpaci, R. H., Dusseau, A. C., Vahdat, A. M., Liu, L. T., Anderson, T. E. and Patterson, D. A.: The Interaction of Parallel and Sequential Workloads on a Network of Workstations, UC Berekeley Technical Report CS-94-838, Computer Science Division, University of California, Berekeley (1994).
11) Burger, D. C., Hyder, R. S., Miller, B. P. and Wood, D. A.: Paging Tradeoffs in Distributed-Shared-Memory Multiprocessors, *Supercomputing'94*, pp. 590–599 (1994).
12) 堀敦史, 石川裕, 小中裕喜, 前田宗則, 友清孝志: 時分割空間分割スケジューリング, 情報処理学会論文誌, Vol. 37, No. 7, pp. 1320–1331 (1996).
13) Franke, H., Pattnaik, P. and Rudolph, L.: Gang Scheduling for Highly Efficient Distributed Multiprocessor Systems, *Frontier'96* (1996).

る．DQT は既に SCore-D に実装されている．