社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

フォールト／リカバリモデルを考慮した耐故障性をもつ MPI フレームワーク Cuckoo FTMPI の提案と評価

實本　英之†　　松岡　　聡†,††

† 東京工業大学
〒 152-8552 東京都目黒区大岡山 2-12-1
†† 国立情報学研究所
〒 101-8430　東京都千代田区一ツ橋 2-1-2
E-mail: †{jitsumo0,matsu}@is.titech.ac.jp

あらまし　並列，広域化に広く用いられる MPI において，耐故障性を備えたものが作られた．しかしそれらは環境ごとに変わるリカバリモデルを考慮した物が少なく，考慮した物も，状況に応じた復旧手段をコード中に記述する必要がありユーザーの負担が多い．本研究では環境に合わせたリカバリモデルを容易に用いることが可能な MPI フレームワーク Cuckoo FTMPI を提案する．リカバリモデルをコンポーネントとして提供することにより，ユーザーはコンポーネントを選択するだけで，環境に最適な耐故障性 MPI を用いることが可能となる．MPICH に Cuckoo FTMPI フレームワークを用いた拡張を行い，さらに 2 種の並列耐故障性コンポーネントを実装し，NPB による性能評価を行った．結果，コンポーネントをアプリケーションに応じて変更することが性能に大きな影響を与えることを示した．
キーワード　耐故障性 MPI，コンポーネントフレームワーク，リカバリモデル

The Proposal and Evaluation of Cuckoo FTMPI: Framework of Fault/Recovery model aware Component-based FTMPI

Hideyuki JITSUMOTO† and Satoshi MATSUOKA†,††

† Tokyo Institute of Technology
2-12-1 Ookayama, Meguro-ku, Tokyo, 152-8552 JAPAN
†† National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, 101-8430 JAPAN
E-mail: †{jitsumo0,matsu}@is.titech.ac.jp

Abstract　Execution of MPI applications on clusters and Grid deployments suffering from node and network failures motivates the use of fault tolerant MPI implementations. Therefore, some fault tolerant MPI was implemented. But, these fault tolerant MPI implementations cannot choose easily appropriate restoration according to the environment. We present Cuckoo FTMPI: Fault/Recovery model aware component framework. Users can get a MPI implementation according to their executing environment by the selection of the components. This paper presents the architecture of Cuckoo FTMPI, its theoretical foundation and the performance of the implementation. Preliminary evaluation using NPB, there's no overhead to use Cuckoo FTMPI on MPICH. And we presented validity of Fault/Recovery model aware component framework.
Key words　Fault Tolerant MPI, Component Framework, Recovery-model-aware

1. はじめに

コモディティクラスタリングシステムやグリッドシステムの発展に伴い，多くの科学技術計算が並列，広域化されている．これらの並列，広域化に用いられるインターフェースの 1 つとして MPI がある．MPI は並列プログラムで使われる汎用通信インターフェースであり，様々な環境に適合した実装を持っている．さらに，物理，化学，生命学といった様々な分野のユー

ザが使用しており，長時間の科学技術計算にも利用されている．これらの要件を考慮すると，理想的な MPI は以下のような物であると考える．

異常終了を起こさない　MPI が用いられる科学技術計算は長時間にわたる．しかし，実行環境となるクラスタやグリッドは様々な要因でノードの信頼性が低い．よって耐故障機能を持つべきである．

簡単に利用できる　MPI は計算機科学以外の分野でも研究のツールとして利用される．このため，手軽にユーザの必要とする並列アプリケーションプログラムを記述できる必要がある．

環境に依存しない　MPI は様々なオペレーティングシステムやハードウェアの上で実行される．また， 様々なソフトウェアアーキテクチャを実装可能である．このため，特定の環境やソフトウェアアーキテクチャに依存しないよう実装する必要がある．

初期の， MPICH [1], LAM などの MPI 実装は，MPI インターフェースを用いて様々な環境で簡単に利用可能だが，耐故障性に関しては配慮されていない．近年のクラスタやグリッドなどの普及により，耐故障性が必須となった結果，LAM/MPI [2] や FT-MPI [3] といった耐故障性 MPI が実用化され始めている．しかしながら，これには以下のような問題がある．

復旧方法が画一的　LAM/MPI や MPICH-V [7] を初めとする多くの耐故障性 MPI は，フォールトが起こったときにどのような復旧を行うかが画一的に決まっている．本来，フォールトの種類や環境毎に適切な復旧方法は変わる．このため従来の耐故障性 MPI では余分なコストがかかってしまう．例として，常に他のノードにプロセスを復旧する復旧方法しか持っていない場合，ノードの故障によるシステムの停止を防ぐことはできるが何らかの原因でプロセスのみが異常終了してしまったときは正常なノードを放棄することになる．

並列耐故障機能の変更や実装が困難　並列プロセスの耐故障機能を実現するためには，チェックポインティングだけでも様々な方法がある．これら耐故障機能のコストは，アプリケーションの性質によって大きく変わる．幾つかの耐故障性 MPI 実装ではこれらが実装に強く結びついており変更が難しい．また，FT-MPI のようにプロセスの生成／破棄といった基本的な API のみを提供して，アプリケーションユーザに耐故障性を実装させる MPI も存在するが，ユーザはアプリケーションの内容を熟知している必要があり，実装は難しい．

特定の OS/HW への依存性が高い　耐故障性 MPI はその実現に，チェックポインタや高信頼ネットワークなどを使用している．これの実装は特定の OS やハードウェアに依存してしまうことが多く MPI に必要とされる多くの環境へ非依存であることを損なう．

以上を解決するため，環境に合わせた復旧方法を容易に用いることが可能な MPI フレームワーク Cuckoo FTMPI を提案する．Cuckoo FTMPI は並列耐故障アルゴリズム，フォールトモデル定義，リカバリプロトコルを組み合わせたリカバリモデル（復旧方法）中心のコンポーネントモデルを持つ．コンポーネントは，動的ライブラリで提供されており，ユーザ，システム管理者，ソフトウェア開発者が，適切なコンポーネントを組み合わせていくことで，環境およびソフトウェアアーキテクチャに最適な耐故障性 MPI を用いることが可能になる．

2. Cuckoo FTMPI フレームワーク

2.1 リカバリモデル

Cuckoo FTMPI フレームワークでは以下の 3 つの機能を組み合わせることによりリカバリモデルを定義する．

並列耐故障アルゴリズム　通常実行時に行う並列耐故障機能の準備，故障発生時に復旧させるプロセスの選定を定義する．並列プロセスの耐故障性を実現するためには通信の一貫性の保証を行わなければならない．この一貫性の保証手段により，選択可能なリカバリプロトコルは変化する．例としてチェックポインティングを上げると，同期が必要な Coordinated Checkpointing は必ず全てのプロセスをリカバリしなければならない．対して，チェックポイントからのロールフォワードを行う Log-based Checkpointing では，正常実行時はメッセージログの作成を行う必要があり，復旧時はログを使用しながら故障プロセスのみのリカバリを行う．

リカバリプロトコル　並列プロセスの構成プロセス毎の実際のリカバリ処理を定義する．Cuckoo FTMPI ではリカバリ処理を 4 つのプロトコルに大別する．

IGNORE　対象プロセスのリカバリを行わず無視する

RESTART　対象プロセスが元々動いていたノードにリカバリする

MIGRATION　対象プロセスを元々動いていたノード以外でリカバリする

TRANSFER　対象プロセスのレプリケーションプロセスをプライマリにする

図 1 はこれらのプロトコルの特徴である．復旧時コストはプロセスイメージの移動を伴う MIGRATION が大きく，プロセス再生処理のみの RESTART，プロセス再生処理も必要ない TRANSFER の順に小さくなる．また必要な冗長ノード数はプロセスレプリケーションを行う TRNSFER が大きく，MIGRATION においても障害が起きるたびに冗長ノードが消費される．故障の程度にあわせて適切なリカバリプロトコルを選択することにより障害復旧のためのコストを押さえることが可能である．例としては，ノード数の大きい環境では TRANSFER を使用することや，Master-Worker 方式のアプリケーションでは IGNORE を選択する等の方針がとれる．

フォールトモデル定義　実行環境のモニタリング情報からどのリカバリプロトコルを選択するか定義する．簡単な例として以下のようなフォールトモデルを挙げる．

Network Fault　冗長化ネットワークの一部の障害，あるいはパケットロス等によるネットワーク性能の著しい低下状態．

Process Fault　並列ジョブを構成するプロセスが何らかの原因により異常終了してしまった状態．

Physical Fault　ハードウェアの障害により，物理的にノードが使えない状態．また，ネットワークが完全に切断されノードが応答不能になった状態

表 1　リカバリプロトコルの性質

プロトコル	IGN.	RES.	MIG.	TRA.
復旧時コスト	無し	中	大	小
冗長ノード数	無し	無し	障害頻度	大
耐プロセス障害	無し	有り	有り	有り
耐ノード障害	無し	無し	有り	有り

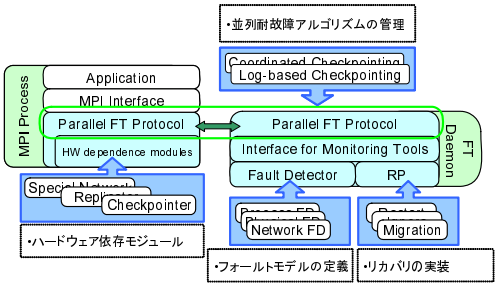

図 1　Cuckoo FTMPI コンポーネント図

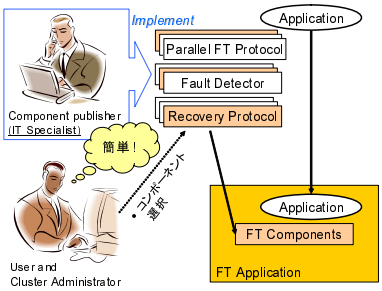

図 2　コンポーネントの利用

また，これらが組み合わされたモデルである Repeated Fault 等も定義可能である．

Repeated Fault　プロセス障害回数が閾値をこえ，ハードウェア障害が起きていることが疑われる状態．

プロセスが Process Fault モデルに従い適切なプロトコルで復旧を行うにもかかわらず，何度も同じフォールトが発生することがある．例をあげるとノードのメモリの故障によりメモリ上のデータが不適正な値になってしまうことによるプロセス障害などがあげられる．この場合，ノード自身は正常動作しているように見えるため，Process Fault モデルと認識される．しかしながら，故障しているメモリ領域を用いるたびにプロセスが予期しない動作をとり，異常終了を繰り返す．実際の定義方法としてはモニタからノード毎の異常終了頻度を取得し，閾値を越えたときには Physical Fault モデルとして扱うというものになる．

2.2　コンポーネント

Cuckoo FTMPI フレームワークの全体図を図 1 に示す．コンポーネントはプログラミング経験を豊富に持つスペシャリスト（コンポーネントパブリッシャ）により制作されるもので，システム管理者，ユーザは各人が必要とするコンポーネントを選択することにより容易に耐故障機能をカスタマイズできる．(図 2)

図中 Parallel FT Protocol (PFTP) コンポーネントは並列耐故障アルゴリズム機能，Recovery Protocol (RP) コンポーネントはリカバリプロトコル機能，Fault Detector (FD) コンポーネントはフォールトモデル定義機能にそれぞれ一致する．また，他のコンポーネントとして，モニタリングツールから環境の情報を得るためのインターフェースとハードウェア依存性の高い高信頼通信やチェックポイントなどをコンポーネントとして利用する機能をもつ．

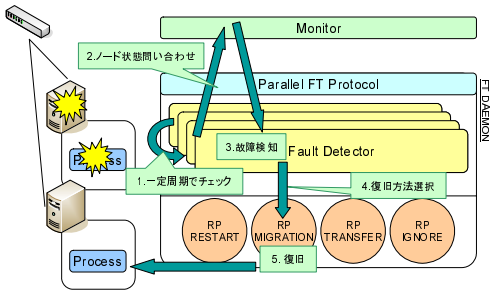

図 3　リカバリ時のコンポーネント実行

リカバリは以下の手順で行われる．(図 3)

（1）　PFTP コンポーネントは定期的に FD コンポーネントのチェック関数を呼び出す．

（2）　FD コンポーネントは外部のモニタに実行環境の情報を問い合わる．

（3）　FD コンポーネントはモニタから受け取った情報が，自らの定義するフォールトモデルに合致するか確認する．

（4）　FD コンポーネントは合致したフォールトモデルに対応する RP コンポーネントを選択する

（5）　PFTP コンポーネントは FD コンポーネントに選択された RP コンポーネントを使用してリカバリ処理を行う．

3.　実　　装

プロトタイプとして MPICH-P4MPD に Cuckoo FTMPI を導入し耐故障 MPI の実装を行った．また，Coordinated Checkpointing，Pessimistic Log-based Checkpointing の 2 種の PFTP コンポーネント，ユーザレベルチェックポインタ ckpt [4] を使用するためのチェックポインタコンポーネントを実装した．各コンポーネントは動的ライブラリとして提供され，MPICH-P4MPD に埋め込まれたエントリポイントからコンポーネントのインターフェース (Cuckoo IF) が呼び出される．コンポーネントの読み込みは設定ファイルに読み込む動的ライブラリを記述することによって行う．

3.1　MPICH へのフレームワークの適応

PICH-P4MPD ではジョブの起動は以下のように行われ，結果，図 4 のようなプロセス間通信が行われる．

（1）　各ノードには MPD が実行されており，リング型に接続されている．

（2）　ユーザの mpirun が mpd にユーザプログラム実行命令を送る

（3）　ユーザプログラム実行命令が mpd 内を伝播しながら，実行したいプロセスにつき一つの mpdman を起動する

（4）　mpdman は隣の rank の mpdman とコネクションを張り，リング型の通信路をつくる

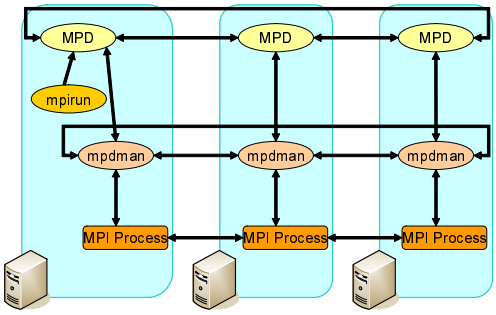

図 4　MPICH-P4MPD 全体図

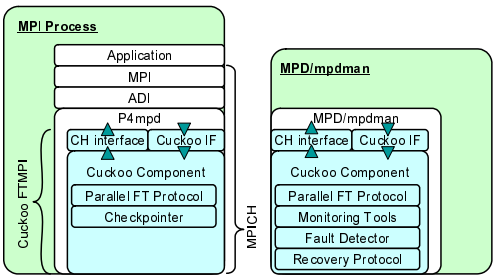

図 5　MPICH-P4MPD によるプロトタイプ実装

（5） mpdman によりユーザプログラムが実行される

これをふまえて，以下の場所に各コンポーネントが追加処理を実行することのできるエントリポイントを設けた．

- MPD の左手，右手，mpdman からの受信部分
- mpdman の左手，右手，MPD，MPIProcess からの受信部分
- MPIProcess の mpdman からの受信部分
- MPIProcess 間の送受信部分

MPIProcess 間の送受信について，MPICH-P4MPD はすべての MPI 通信を P4 ライブラリの関数 p4_recv，p4_sendx，p4_available_messages の 3 つをのみを使って実現している．そこで MPI 通信の内容を管理するためにそれぞれの関数のラッパーを動的ライブラリとして提供できるように変更した．さらに mpdman にタイマを設定し定期的なイベントを行うためのエントリポイントを設置した．

図 5 はプロトタイプ実装のコンポーネント図である．MPICH-P4MPD は Cuckoo IF を利用してコンポーネントを利用する．また，コンポーネントは CH Interface を使うことにより P4MPD のもつランクやジョブサイズといった情報を利用したり，他の MPIProcess や MPD，mpdman と通信することができる．コンポーネントの実装は図 6 のようになる．この例は PFTP コンポーネントの p4_sendx ラッパの部分である．P4MPD は Cuckoo IF を用いて PFTP コンポーネントに定義されたラッパ RB_send を呼び出す．RB_send はメッセージ／イベントのロギングといった追加処理を行い，CH Interface を用いて p4_sendx を実行する構造になっている．

3.2　Parallel FT Protocol コンポーネントの実装

Cuckoo FTMPI フレームワークによるのコンポーネントの変更による環境，アプリケーションへの柔軟な適応を検証するために Pessimistic logging Checkpointing， Coordinated Checkpointing [5] の 2 つの PFTP コンポーネント実装を行った．

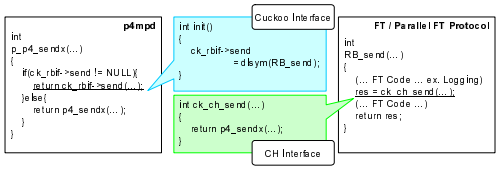

図 6　コンポーネントの実装例

3.2.1　Pessimistic logging Checkpointing

チェックポイントからの非決定イベントをロギングしておきリスタート後ログに従いイベントを再発生させる．通信の度にロギングコストがかかることから通信頻度の高いアプリケーションには向いていないが，MPI 構成プロセスの同期が必要ないためプロセス数の大きなアプリケーションに向いている．実装は大きく分けてロギング，リカバリ，ログのガベージコレクションからなる．

ロギング　イベントログとメッセージログの 2 つを保存する．イベントログは送信元でのログバージョン (s)，送信先でログバージョン (r)，のイベントの宛先 (dst) と送信元 (src)，イベントの結果値 (v) からなり，MPI Process において p4_sendx, p4_recv, p4_available_messages が呼ばれるたびにその結果を保存する．同時にログサーバに対してログを送信する．また，Lamport 順序を保つためにメッセージ送信の直前に全てのイベントログがログサーバに保存されていることを保証する ACK 処理を行う．メッセージログは送信元でのログバージョン (s) とメッセージの宛先 (dst) と送信元 (src)，メッセージ本体 (msg) からなり，p4_sendx が呼ばれるたびに MPI Process に保存される．

リカバリ　RP コンポーネント (もしくは手動) によって再開されたチェックポイントはイベントログサーバからログを回収する．再実行中，ログを参照しメッセージの再送が必要な場合は相手プロセスに mpdman を通じて要求を送る．要求を受け取った相手 mpdman は MPIProcess に対してシグナルを送り，MPIProcess はシグナルハンドラ内から必要なメッセージを再送する．

ガベージコレクション　イベントログはチェックポインティング開始直前に全て破棄．メッセージログはチェックポイントを行ったプロセスが送信元プロセスにチェックポイントに含まれたメッセージのログバージョンを伝え，それ以前のメッセージを破棄する．

3.3　Coordinated Checkpointing

すべてのプロセスが同期して通信の一貫性を取り，チェックポインティングを行う．このため，各構成プロセスが任意のタイミングでチェックポインティングを行うことはできない．MPI 構成プロセスの同期処理を行うためにプロセス数が大きいアプリケーションには向いていない．しかし，チェックポイント時以外に行うロギングなどの処理がないため通信頻度等は性能に影響しない．本研究では以下のように通信の一貫性を保証している．

同期　チェックポイントが開始されると，プロセスは接続され

表 2 評価環境

CPU	AMD Optelon Processor 242 (1.6GHz)×2
Memory	2GB
Network	1000base-T
OS	Linux 2.6.14

表 3 NPB: MPICH-P4MPD VS Prototype

	Original(Mops)	Prototype(Mops)	deg. ratio (%)
CG	1034.11	999.47	3.3%
MG	2744.32	2732.25	0.43%
EP	44.67	44.64	0.067%
IS	35.85	35.21	1.8%

ている全てのプロセス間ソケットに Drained メッセージを送る．接続されている全てのソケットからの Drained メッセージを受け取ったプロセスはチェックポイントを開始する．これにより in-flight なメッセージが無いことが保証される．また，チェックポイントを終了したプロセスは，全てのプロセスがチェックポイントを終了するまで他プロセスへの送信を行わない．

4. 性能評価

4.1 評価環境

評価環境として，東京工業大学松岡研究室の PrestoIII クラスタを用いた．各ノードは表 2 の構成である．

4.2 MPICH-P4MPD の拡張によるオーバーヘッド測定

Cuckoo FTMPI フレームワークを適用するために MPICH-P4MPD に行ったエントリポイントの追加やタイマの設定が及ぼす性能低下を見るために，コンポーネントを全てはずしたプロトタイプ実装とオリジナルの MPICH-P4MPD との性能比較を行った．

- NPB CLASS A CG/MG/IS/EP
- 8 プロセス/8 ノード

結果を表 4.2 に示す．性能低下は最大で 3.3%と小さいもので，MPICH-P4MPD の拡張による性能低下はほとんど無いことが確認できる．

4.3 コンポーネント変更によるアプリケーションへの適応確認

環境やアプリケーションの性質に対して適切なコンポーネントを使用することにより，効率の良い実行ができることを確認するために，2 種の PFTP コンポーネントを用意し，NPB-CG/EP を対象として性能比較を行った．NPG-CG はプロセス間の通信頻度が大きいベンチマークである．対して EP は通信がほとんど無い．

4.3.1 通常実行時オーバーヘッド

通常実行時のオーバーヘッドを確認するためにチェックポインティングを行わずに PFTP コンポーネントの性能比較を行った．

- NPB CLASS A CG/EP
- 2, 4, 8, 16, 32 プロセス/ノード毎に 1 プロセス
- CC: Coordinated Checkpointing
- PML: Pessimistic Log-based Checkpointing

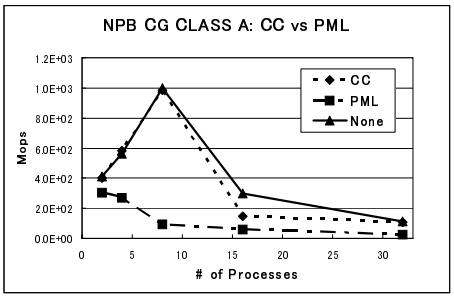

図 7 NPB CG: PFTP の違いによる比較

- None: チェックポインタ無し
- チェックポイントはおこなっていない

結果を図 7,8 に示す．通信毎にメッセージログを行う PML では CG において顕著な性能低下が見られるが，EP においては CC との性能差は最大で 5%程度となった．PML が 1 プロセスずつリカバリできることも考えると，EP のような通信の少ないアルゴリズムでは PML を使うほうが優れている．これより，アプリケーション等の性質に対して適切なコンポーネントの使用が有効であることがわかる．

4.3.2 チェックポイント時オーバーヘッド

チェックポインティングのオーバーヘッドを確認するために，実行中必ず 1 回チェックポインティングを行うよう設定して PFTP コンポーネントの性能解析を行った．

- NPB CLASS A CG
- 2, 4, 8, 16, 32 プロセス/ノード毎に 1 プロセス
- CC: Coordinated Checkpointing
  - MSYNC: メッセージドレインのための同期時間
  - CKPT: プロセスイメージのファイル化にかかった時間
- PML: Pessimistic Log-based Checkpointing
  - ELDUMP:イベントログの破棄にかかる時間
  - CKPT:プロセスイメージのファイル化にかかった時間
  - GC:他プロセスへのメッセージログの破棄命令送信時間

結果を表 4 に示す．CC は不具合により 8 プロセス以上の評価ができなかった．しかし，2 プロセスから 4 プロセスで同期時間が増加することが確認できた．対して PML はプロセス数の増加に対して ELDUMP, CKPT, GC ともに相関関係が見られない．本実験ではプロセス数が少ないために CKPT に対して ELDUMP の影響がとても小さいくなってしまった．同期時間の増加が致命的なものであるかを確認するにはメガスケールでの追実験を行う必要がある．また，プロセス数や通信頻度だけでなく，メッセージサイズといったその他のパラメータに関してもより詳細に評価を行う必要がある．

5. 関連研究

5.1 LAM/MPI，Open MPI

LAM/MPI，Open MPI [6] は MPI 実装の一つとして LAM を用いたものである．LAM/MPI は SSI (System Service Interface) と呼ばれるコンポーネントをもち，起動手順，Peer-to-peer 通信アルゴリズム，集団通信アルゴリズム，チェックポインティ

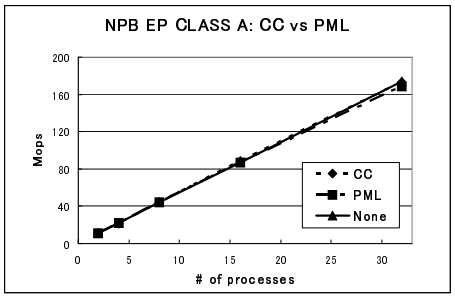

図 8　NPB CG: PFTP の違いによる比較

表 4　NPB CG: チェックポインティング時間詳細 (秒)

CC		
NODE	SYNC	CKPT
2	4.1×10^{-4}	1.15
4	7.4×10^{-4}	1.29

PMLWS			
NODE	ELDUMP	CKPT	GC
2	1.9×10^{-5}	1.82	8.1×10^{-5}
4	1.5×10^{-5}	1.77	5.0×10^{-4}
8	1.4×10^{-5}	1.51	1.0×10^{-4}
16	1.6×10^{-5}	1.34	1.0×10^{-4}
32	1.7×10^{-5}	2.74	1.7×10^{-4}

ング／リスタートなどを追加・変更できる．耐故障をはじめ殆どが，LAM/MPI で用いられているコンポーネント分けと同様である．しかし，チェックポインティング／リスタート SSI には自動リカバリのための故障検知機構の仕様は含まれていない．

5.2　MPICH-V Project

MPICH-V [7] は本研究と同様，並列チェックポインティングのアルゴリズムを変更できる耐故障 MPI である．全ての通信はプロセスから一度メッセージロギングデーモンを通過するため，レイテンシが高くなる．本研究との差異は故障検知機構，自動リカバリを持っているが，完全に固定されてしまっているため，フォールトモデルにあわせたリカバリ手法の変更はできない所にある．

5.3　FT-MPI

FT-MPI [3] はユーザ透過な耐故障性を提供しない．ユーザが MPI プログラムに直接，故障への対処を記述することにより耐故障性を発揮する．通常の MPI 規格では，MPI ジョブ中のひとつのプロセスがダウンしたり，通信に失敗が起こると，MPI ジョブ全体がクラッシュする．FT-MPI はコミュニケータおよびプロセスの状態拡張しており，コミュニケータのエラーの検知，および障害への柔軟な対処方法をユーザに提供する．プログラマがエラー処理に関するコードを追加しなければならず，非常に負荷が高い．

6.　まとめと今後の課題

フォールト／リカバリモデルを考慮した耐故障性をもつ MPI フレームワーク Cuckoo FTMPI を提案した．また Cuckoo FTMPI フレームワークを実際に MPICH-P4MPD に導入し，導入オーバーヘッドがほとんど無いことを確認した．NPB CG/EP において 2 種の PFTP コンポーネントについて性能比較を行い，アプリケーションの性質に適したコンポーネントを使うことにより耐故障機能のオーバーヘッドを削減できることを確認した．

今後の課題として以下を行う．

リカバリモデルの動的変更　同じ環境やアプリケーションを用いても，実行時間やそれまでの故障率により最適なリカバリモデルを考慮することが可能である．簡単な例としては一定時間故障が起きなければ徐々にチェックポインティングの周期を長くすることなどである．また，アプリケーションの性質やハードウェアの構成を検知し，自動的な最適リカバリモデルの構成なども行っていく．

並列チェックポインティングアルゴリズムの評価　様々な並列チェックポインティングアルゴリズムが提案されているが，それぞれのコストがどのようなパラメータによって変化するかは明確になっていない．このパラメータを分析することにより，どのようなアプリケーションにどのチェックポインティングアルゴリズムが最適なのかを検証する．

謝　　辞

本研究の一部は科学技術振興事業団・戦略的創造研究「低電力化とモデリング技術によるメガスケールコンピューティング」による．

文　　献

[1] W. Gropp, E. Lusk, N. Doss and A. Skjellum: “High-performance, portable implementation of the MPI Message Passing Interface Standard”, Parallel Computing, **22**, 6, pp. 789–828 (1996).

[2] J. M. Squyres and A. Lumsdaine: “A Component Architecture for LAM/MPI”, Proceedings, 10th European PVM/MPI Users’ Group Meeting, No. 2840 in Lecture Notes in Computer Science, Venice, Italy, Springer-Verlag, pp. 379–387 (2003).

[3] G. Fagg and a Dongarra: “FT-MPI: Faulttolerant mpi,supporting dynamic applications in a dynamic world” (2000). Euro PVM/MPI User’s Group Meeting 2000 ,Springer-Verilag, Berlin, Germany, 2000, pp.346-353.

[4] V. C. Zandy: “ckpt: A process checkpoint library” (2002). http://www.cs.wisc.edu/˜zandy/ckpt.

[5] E. Elnozahy, D. Johnson and Y. Wang: “A survey of rollback-recovery protocols in message-passing systems” (2002). ACM Computing Surveys, 34:3, September 2002, pp. 375-408.

[6] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham and T. S. Woodall: “Open MPI: Goals, concept, and design of a next generation MPI implementation”, Proceedings, 11th European PVM/MPI Users’ Group Meeting, Budapest, Hungary, pp. 97–104 (2004).

[7] A. Bouteiller, T. Herault, G. Krawezik, P. Lemarinier and F. Cappello: “Mpich-v: a multiprotocol fault tolerant mpi”, To appear in International Journal of High Performance Computing and Applications. (2005).