Vol.2011-SLDM-149 No.17
Vol.2011-EMB-20 No.17
2011/3/18

MCAPIを用いた組込み向け耐故障分散共有メモリの実装

鈴木良平[†1] 三浦信一[†2] 李珍泌[†1]
塙敏博[†1,†2] 朴泰祐[†1,†2] 佐藤三久[†1,†2]

航空機や鉄道車両，産業用機器や医療用機器などに用いられている組込みシステムは，システム停止時の影響が非常に大きく，その耐故障性を極力高める必要がある．そのために，マルチノードシステム上の各ノードのメモリに各チェックポインティングデータをバックアップとして保存することで耐故障性を高めるという方法がある．そこで，本研究ではチェックポイント/リスタート機構が実装されたソフトウェア耐故障性分散共有メモリ SCASH-FT を用いる．さらに，SCASH-FT をマルチノード上で動作させるために，MCAPI の一実装である XMCAPI に着目し，SCASH-FT のプロセス間通信 API として用いる．そして，組込みシステム向け耐故障ソフトウェア分散共有メモリ SCASH/MCAPI を提案・実装する．予備評価として SCASH/MCAPI を 2 ノードのシステム上に実装し，二次元ラプラス方程式を解き，システムの動作の正しさを確認した．また，今回の実装ではベースとなる MCAPI 実装として汎用マルチノードにおいて TCP/IP 上で動作する XMCAPI を用いているため，TCP/IP をそのまま用いる SCASH-FT に比べオーバヘッドが大きいことがわかった．

Software Distributed Shared Memory for Embedded System by MCAPI

Ryohei Suzuki,[†1] Shinichi Miura,[†2] Jinpil Lee,[†1]
Toshihiro Hanawa,[†1,†2] Taisuke Boku[†1,†2]
and Mitsuhisa Sato [†1,†2]

Embedded systems implemented in aircrafts, railway carriers, industrial equipment or medical equipment require high reliability to avoid serious system faults. Such a fault tolerant technology should be also implemented on multi-node systems. One of the ideas to make a multi-node fault tolerant system is to provide distributed shared memory system on it to keep back-up copies of memory image among these nodes with each other to tolerate any node fault. SCASH-FT was such a fault tolerant software DSM based on TCP/IP, and we will port it for more appropriate embedded communication system, MCAPI. In this paper, the prototype of this new software DSM named SCASH/MCAPI is designed, implemented and evaluated on two-node PC cluster. It is confirmed that the system properly operates to solve 2D Laplace Equation with creation of check-point data to prepare further node failure. Since this prototype is implemented on a portable MCAPI library named XMCAPI which is available on any general PC servers with TCP/IP as the basic communication layer, the performance is lower than SCASH-FT with direct TCP/IP.

1. はじめに

近年, デジタル家電や携帯電話，自動車など特定の用途に向けて設計された組込みシステムが普及してきている．年々，これらの組込みシステムでは必要とされるデータ処理量が増加し，より高性能なプロセッサが求められている．そこで，汎用 PC で利用されてきたマルチコアプロセッサ技術が組込みシステムのプロセッサにも広く応用され，高性能化と消費電力削減の両立を実現している．しかし，現在の組込みシステムのプロセッサコア数はコストやチップから発生する熱の冷却を考えると 2〜4 個が限界のため，より高い性能を得るには，複数のマルチコアプロセッサチップをネットワークで接続した，マルチノードシステムにする必要がある．マルチノードシステムは，高い処理性能を持つが，シングルノードシステムよりも複雑になるため故障確率が上昇する．また，マルチノードシステムが用いられている航空機や鉄道車両，産業用機器や医療用機器などは，停止すると大きな損害を与える可能性がある．そのため，マルチノードシステムには高い耐故障性が求められている．

そこで我々は，システム上に冗長ノードをスタンバイさせ，故障時にこれを代替ノードとして用いるという耐故障手法を，ソフトウェア分散共有メモリ（Software Distributed Shared Memory, 以下 SDSM）のフレームワーク上で実現する，組込み向け耐故障 SDSM として SCASH/MCAPI を提案する．これまで我々は，耐故障 SDSM を実現する SCASH-FT[1)] を開発してきた．これは SDSM において，ユーザが指定する適当なタイミングでシステム上のノードが相互のメモリイメージを複製することにより，あるノードに故障が発生した場合

†1 筑波大学大学院システム情報工学研究科
University of Tsukuba, Graduate School of Systems and Information Engineering
†2 筑波大学計算科学研究センター
University of Tsukuba, Center for Computational Sciences

Vol.2011-SLDM-149 No.17
Vol.2011-EMB-20 No.17
2011/3/18

にバックアップノードへのメモリイメージを復旧支援するシステムである．SCASH-FT は汎用システムに用いるため，TCP/IP 通信を用いて実装されている．SCASH/MCAPI は SCASH-FT の実装を基本とし，組込みシステムでの利用を意識し，今後組込みシステムにおいて標準通信ライブラリの一つになると見込まれる MCAPI (Multi-core Communication API)[2] を用いて実装する．MCAPI とは，Multicore Association[3] によって仕様策定された，通信オーバヘッドが少なく，メモリフットプリントが小さいという特徴を持ったマルチコアプロセッサ向けのコア間通信 API である．そのため，SCASH-FT の通信部分を MCAPI に置き換えれば，様々な組込みシステム上での動作が期待できる．

2. SCASH/MCAPI の概要

本研究の目的は，マルチノードシステム上でソフトウェアによって耐故障性を高めることである．マルチノードシステムは高性能・低消費電力な組込みシステムの実現に有効であるが，その複雑性から，故障発生の可能性がシングルノードシステムよりも格段に高まる．従って，高い信頼性が要求される組込みシステムにおいては，耐故障性を持つマルチノード制御ソフトウェアが不可欠であると考えられる．そこでチェックポインティング/リスタート機能を持つ SDSM である SCASH-FT をコア間通信 API である MCAPI を用いて実装する．これにより，高性能かつ高信頼性を持つマルチノード組込みシステムが実現できる．また，組込みシステム向けの標準的な通信 API として期待される MCAPI を用いるため，様々な組込みシステムへの移植が期待できる．本研究では，MCAPI で実装した SCASH-FT を SCASH/MCAPI と示す．次に，本研究で用いた SDSM である SCASH, さらにチェックポインティング・リスタート機構を追加した SCASH-FT について述べ，コア間通信機構 MCAPI について説明する．

2.1 SCASH

SDSM とは，分散メモリ型並列計算機の上でソフトウェア技術を用いて仮想的に共有メモリを実現したものである．代表的な SDSM の実装には，TreadMarks[4]，JIAJIA[5]，SCASH[6] などがある．SDSM はハードウェアによって実装された分散共有メモリに比べ処理能力が劣るが，ソフトウェアである利点を活かし，システムのアルゴリズムの変更や新しい機構の追加が比較的自由であるなど拡張性に優れている．

SCASH は，PC クラスタコンソーシアムによって開発されたクラスタシステムソフトウェア SCore[6] が提供する SDSM である．SCASH は，ユーザレベルライブラリとして提供されており，各種 API の使用し，仮想的な共有メモリを分散メモリ上で実現する．ユーザが明示的にコード中に API を挿入することで，共有メモリの割り当て，メモリの同期などの一貫性制御を行う．

2.2 SCASH-FT

SCASH-FT[1] は，SCASH 上でチェックポインティング/リスタート機能を実装した SDSM である．通常のチェックポインティングは，チェックポインティングに関わるデータをノード上のディスクに保存する．リスタートの際には，ノード上に保存されたチェックポインティングデータをメモリなどに展開し，実行を再開する．SCASH-FT の特徴として，チェックポインティングデータをノードのディスクではなくメモリ上に保存する．これにより，ディスクがないシステムでもチェックポインティトを行える．また，SDSM の機構に従うことで，チェックポインティングデータを他のノードの仮想共有メモリにバックアップとして保存できるため，チェックポインティングデータの冗長化を可能にしている．このように冗長化されたチェックポインティングデータを参照することで，どのようなノードが故障した場合においてもリスタートすることができる．

図 1, **図 2** に SCASH-FT におけるチェックポインティング/リスタート機能の概念を示す．図 1 に示す通常状態において，並列プログラム実行中に，各ノードのデータに対するチェックポインティングを行う．図 2 のように，ノード 2 が故障した場合，予備ノードであるノード 4 にノード 3 が保持するノード 2 のチェックポインティングデータを転送する．その後，ノード 4 上でノード 2 のデータを展開し，リスタートすることによりシステム全体が再起動できる．以上のように，メモリベースのチェックポインティング/リスタートを行うことで，常にシステムが稼動し続けることができ，また一般的に行われているようなディスクベースのチェックポインティングではないため，復旧時の即応性に優れ，リアルタイム性の高い耐故障性を実現できる．

2.3 MCAPI

マルチコアチップ上のコア間通信 API として，Multicore Association[3] によって Multi-core Communications API[2] (以下，MCAPI) が提案されている．本研究では，SCASH-FT を組込みシステムに実装するため，組込みシステム向けの標準的な通信 API になると期待される MCAPI を用いる．

MCAPI は，マルチコアプロセッサのチップ内におけるコア間通信用に仕様策定された通信 API である．ただし，MCAPI では，コア間通信をそれらのコア間で共有されるアドレス空間上のメモリ参照等で行うことを記述するのではなく，コア間通信を分散メモリにおけるメッセージパッシング型の通信として記述する．MCAPI では，コア間通信を前提とし，

Vol.2011-SLDM-149 No.17
Vol.2011-EMB-20 No.17
2011/3/18

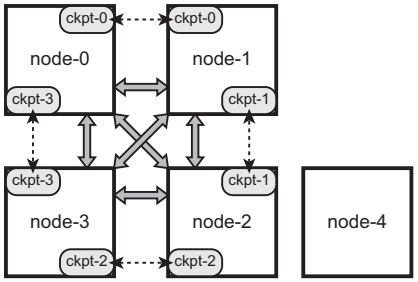

図 1 耐故障 SDSM のイメージ (故障前)

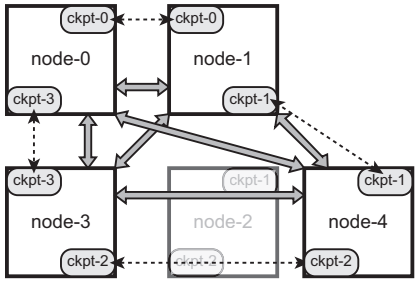

図 2 耐故障 SDSM のイメージ (故障後)

表 1 MCAPI の通信方式

Type	Semantics
Message	コネクションレス指向のデータグラム通信，パケット優先度の設定可
Packet Channel	不定長のデータに対する FIFO 型ストリーム通信
Scalar Channel	特定の整数型 (8~64 bit) のデータ 1 要素に対する FIFO 型ストリーム通信

通信オーバヘッドが少なく，メモリフットプリントの小さいシステム構築が可能な API 体系となっている．また，UNIX 環境における Socket API や高性能並列計算機環境に用いられる Message Passing Interface[7] と非常に近い API 体系でありながら，通信コストを小さくした非常に軽い通信が期待できる．MCAPI では，エンドポイントと呼ばれる通信端を用いて通信を行う．**表 1** に MCAPI が提供する通信方式を示す．MCAPI の規格では，コネクションレス指向のデータグラム通信である Message 型と， コネクション指向の FIFO 型ストリーム通信である Channel 型が存在する．Channel 型は，更に 2 つの種類に分けられ，扱うデータが不定長であれば Packet Channel 型， 特定の整数型のデータ 1 要素に対する通信であれば Scalar Channel 型が用いられる．

3. SCASH/MCAPI の設計と実装

3.1 設 計 方 針

本研究では，SCASH-FT の通信部分に MCAPI を適用し組込みシステム向け耐故障分散共有メモリをユーザに提供する．ただし，既存の SCASH-FT は汎用ネットワークである Ethernet を想定し，socket インタフェースを用いた TCP/IP 実装である．SCASH/MCAPI では，基本的に SCASH-FT で用いられている TCP/IP に基づく通信を MCAPI の適当な API で置き換え，MCAPI 環境を提供するマルチノードシステム上で耐故障 SDSM を実現する．

3.2 SCASH-FT の構成

SCASH-FT は，通信レイヤに TCP ソケットを用いている．UNIX における TCP ソケットライブラリはコネクションの管理が容易であり，他の UNIX システムコードと併用することにより複数の通信の監視を容易にできる．さらに，TCP は広く普及している Ethernet で動作するため，多くのクラスタシステムで利用できる．

SCASH-FT では，ユーザプログラムの計算処理と平行して，非同期に通信を処理するために計算実行用スレッドに加えて通信処理スレッドを起動する．ただし，この通信処理スレッドは，実際には受信処理のみを行うため,「受信スレッド」と呼ばれる．受信スレッド一つに対して計算スレッドはノード内のコア数に応じて複数生成される可能性がある．基本的にメッセージの送信は計算スレッドが行い，メッセージの受信は受信スレッドが行う．受信スレッドは受信したメッセージの種類に従って相手ノードの計算スレッドのリクエストを処理する．通常は計算スレッドからのリクエストは，同期，メモリアドレスの交換などのローカルな処理であるが，リモートメモリアクセス要求の場合にのみ計算スレッドへ必要とされるメモリデータを返信する．

3.3 TCP と MCAPI の差異

SCASH-FT 内で行われる通信は，リモートページの交換や，チェックポインティングデータの複製に伴う通信である．SCASH/MCAPI では，これらの通信を MCAPI による通信に置き換える．

TCP と異なり，MCAPI では表 1 に示した複数の通信手段が提供されている．これらの通信手段は目的に応じて選択される．例えば，Message 通信はコネクションレス通信で簡単に利用可能であるが，メッセージサイズの制限や，通信相手の判断が必要になる．一方 Channel 通信は，通信開始までの手続きは複雑であるが，コネクション確立後は，接続先の判断が不要である．Channel 通信は扱うメッセージサイズが不定長なら Packet Channel 通信，固定サイズの整数型であれば Scalar Channel 通信を利用する．Channel 通信は，汎用的に使用できる Message 通信よりも通信性能の最適化が期待できる．

3.4 SCASH/MCAPI の設計

SCASH-FT は TCP/IP を用いた実装であるため，TCP に近いストリーム型通信を提供する Channel 通信ならばポーティングが容易だと考えられる．さらに，SCASH-FT で実

Vol.2011-SLDM-149 No.17
Vol.2011-EMB-20 No.17
2011/3/18

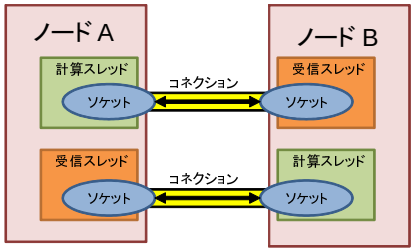

図 3 SCASH-FT

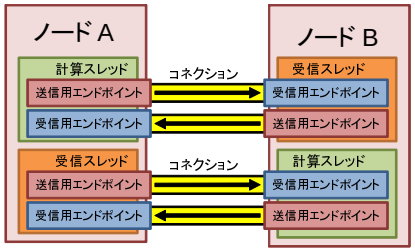

図 4 SCASH/MCAPI

際の通信に用いられるデータサイズは不定長のため，その意味でも今回の実装では Packet Channel 方式を用いる．次に，TCP をどのように MCAPI に置き換えるか簡単に説明する．

通 信 手 順

TCP を用いる socket API は，クライアント/サーバモデルであり，2 ノード間の接続手順が異なる．しかし，接続確立後に作られる識別子 (ソケットファイルディスクリプタ) は対称形の双方向通信を可能にする．TCP と同様に Packet Channel 通信は，2 ノード間の通信確立手順が異なる．それに加え，接続確立後につくられる識別子 (以後，チャンネルハンドラ) は片方向通信を提供し，送信側と受信側を区別する．TCP と同様に双方向通信が必要な場合は，送信用，受信用の 2 つのチャンネルハンドラが必要になる．

チャンネルオープン

チャンネルハンドラの接続には，事前に通信端の準備が必要になる．この通信端の準備以前に，チャンネルハンドラの接続を試みた場合，エラーとなる．そこで本実装では，2 ノード間の通信端の準備完了を保証するために，チャンネルオープンが必要ない Message 通信を用いて，2 ノード間で同期する．

イベント待ち

SCASH-FT の受信スレッドはシステムコール *select()* を用いて複数の受信ソケットを監視しているが，MCAPI のチャンネルハンドラは *select()* を用いて監視できない．そこで，MCAPI で複数チャンネルの通信イベント待ちを行うために用意されている *mcapi_wait_any()* を用いる．

SCASH-FT と SCASH/MCAPI を**図 3** と**図 4** を用いて比較する．SCASH-FT は図 3 のように各ノード間の計算スレッドと受信スレッドを一つのソケットで接続すれば通信可能である．しかし，SCASH/MCAPI の場合，図 4 のように各ノード間の計算スレッドと受信スレッドは 2 つのハンドラを作成しなければならない．そのため，SCASH/MCAPI のハンドラ数は，SCASH-FT のソケット数の 2 倍になる．そして，図 4 のように，各コネクションを送信用，受信用に割り当てる必要がある．

表 2 評価環境

Item	Specification
OS	Linux Kernel 2.6.32.23
CPU	Intel Xeon CPU E3110 3.00GHz dual core
memory	DDR2-800 8 Gbyte
NIC	Broardcom BCM 5721 Gigabit Ethernet

ノード間を接続後は，SDSM である SCASH-FT の動作に従う．基本的に，ローカルノードの計算スレッドがリモートノードの受信スレッドにメッセージを送信することで，同期やチェックポインティングデータを交換する．また，ローカルノードがリモートノードで管理されるページにアクセスする場合にはローカルノードの計算スレッドがリモートノードの受信スレッドにそのページを要求する．そして，要求を受けたページを受信スレッドが計算スレッドに転送する．

4. XMCAPI

現状の MCAPI におけるリファレンスではマルチノードに対応していないが，マルチノード環境に対応した MCAPI 実装として XMCAPI (eXtended MCAPI)[9] が開発されている．XMCAPI は，MCAPI によって記述されたアプリケーションの開発・デバッグ・検証を支援するための汎用的な MCAPI 環境である．現状の XMCAPI は TCP/IP を用いて実装されている．従って，直接 TCP/IP を用いることと比較してオーバヘッドが存在する．しかし，標準的な MCAPI 環境を提供するため，他の MCAPI 環境においてもプログラムの移植が用意になる．このような理由から，今回は SCASH/MCAPI を XMCAPI 上で開発，評価した．

5. 評　　価

前節までの議論を踏まえて，SCASH/MCAPI を実装した．本節では，実装した SCASH/MCAPI の動作確認と，その評価を行う．

評価環境として，**表 2** に示すノードを 2 台用いる． これらのノード間の通信ライブラリとして XMCAPI を用い，その上で SCASH/MCAPI を動作させた．

Vol.2011-SLDM-149 No.17
Vol.2011-EMB-20 No.17
2011/3/18

```
1  iteration = ITR;                                    9     }
2  do {                                               10    }
3   for ( y = y_from ; y < y_to ; y++ ) {             11    temp = uu;
4    for ( x = 1 ; x <= XSIZE ; x++ ) {               12    uu = u;
5     uu[ y * XS + x ]=( u [ (y - 1) * XS + x ]       13    u = temp;
6                      + u [ (y + 1) * XS + x ]       14    scash_barrier( iteration );
7                      + u [ y * XS + x - 1 ]         15   }while ( --iteration );
8                      + u [ y * XS + x + 1 ] )/4.0;
```

図 5 LAPLACE

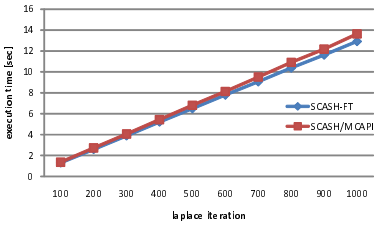

図 6 laplace 実行時間（SCASH-FT と SCASH/MCAPI の比較）

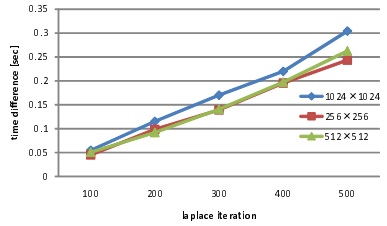

図 7 データ配列サイズを変えた場合の SCASH/MCAPI における処理時間の差

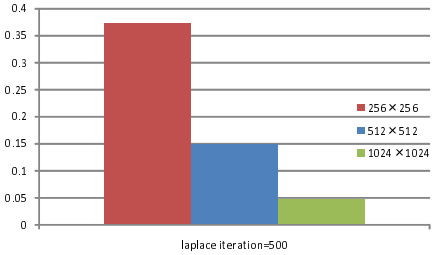

図 8 通信効率

5.1 評価内容

SCASH/MCAPI の動作確認を行い，さらに SCASH-FT と SCASH/MCAPI の性能を比較する．この両方の性能を評価するためのアプリケーションとして，二次元ラプラス方程式 (以下，laplace) を用いる．laplace は二次元ラプラス方程式を差分化し，陽解法による反復計算を実行する．**図 5** に laplace のカーネル部分のソースコードを示す．

この laplace を並列化するために，配列を分散化し，図 5 に示すデータ配列 u と uu を各ノードに割り当てる．iteration 回数を変化させながら SCASH-FT と SCASH/MCAPI 環境における laplace の計算時間の比較評価を行う．この時，動作確認のために，SCASH/MCAPI の計算結果が SCASH-FT の計算結果と同じ値か確認する．本研究では，まだ SCASH/MCAPI におけるチェックポインティング/リスタート機構が完成していないため，今回行う性能評価では，チェックポインティングを行わず，バリア同期のみの性能評価とする．また，今回の評価は全て 2 ノード間での評価のみとする．

本研究では XMCAPI/IP を用いて SCASH/MCAPI を評価するため，TCP/IP を直接用いる SCASH-FT と比較して，性能低下が発生すると考えられる．現在の XMCAPI/IP は，基本的な通信では TCP/IP を使用しているため，通信性能は基本となる TCP/IP の性能に依存する．また，XMCAPI は非同期通信をサポートするためにスレッドを使用しているため，通信遅延が増加する．そこで，laplace による評価に加え，XMCAPI による通信のオーバヘッドを評価する．評価方法として，制御メッセージの転送に必要な時間とページ要求とページ転送に必要な時間を測定する．

5.2 結果と考察

配列サイズを 1024×1024 とした時の各 iteration 回数における laplace の計算時間を**図 6** に示す．

また，配列サイズを変化させた場合の SCASH-FT と SCASH/MCAPI の laplace の計算時間の差を**図 7** に示す．図 6 によると，SCASH-FT と比べ，SCASH/MCAPI の処理時間は多少大きいことが確認できる．加えて，図 7 より，iteration 回数を増す毎にその差は大きくなることが確認できる．また，データ配列サイズが大きくなるにつれ，通信時間の差が大きくなることもわかる．そこで，XMCAPI の通信効率を測定するために，次式を用いた計算時間全体に占める差の割合を**図 8** に示す．

$$\frac{(calc\ time\ on\ SCASH/MCAPI) - (calc\ time\ on\ SCASH\text{-}FT)}{(calc\ time\ on\ SCASH\text{-}FT)} \quad (1)$$

データ配列サイズが大きくなるほど，通信効率が良くなることがわかる．これは，データ配列サイズの大きくなるほど，通信性能差を無視できるようになることを示している．

次に XMCAPI の通信オーバヘッドを調べた．評価方法は制御メッセージの転送に必要な

Vol.2011-SLDM-149 No.17
Vol.2011-EMB-20 No.17
2011/3/18

表 3 通信オーバヘッド

	TCP	XMCAPI
制御メッセージ (24 byte)	49.0 μsec	83.3 μsec
ページ転送を伴う制御メッセージ (4 kbyte)	173.3 μsec	237.2 μsec

時間とページ要求とページ転送に必要な時間を評価した．結果を**表 3** に示す．

結果により，XMCAPI を TCP/IP 上で実装している通信オーバヘッドは TCP/IP の通信時間のオーダーに匹敵するほど大きいことがわかる．現状の XMCAPI はチューニングが不十分であり，このオーバヘッドは決して小さくはない．しかし，本研究では SCASH/MCAPI の開発環境として XMCAPI を用いており，実応用ではよりシステムに特化されチューニングされた MCAPI 環境が期待できるため，今回の性能評価はあくまで参考値である．以上の予備評価より，SCASH/MCAPI が 2 ノード上で正しく動作することを確認した．

6. おわりに

マルチチップ/マルチコア間通信 API である MCAPI を利用して，SCASH-FT に基づく組込みシステム向け耐故障 SDSM である SCASH/MCAPI をプロトタイプ実装した．今回，2 ノード間で二次元ラプラス方程式を用い，TCP 実装である SCASH-FT と SCASH/MCAPI の動作を比較し，検証した．その結果，SCASH/MCAPI の laplace の計算結果が SCASH-FT と一致することから正しい動作を行っていることがわかった．また，評価結果からわかるように，純粋な TCP 実装である SCASH-FT よりも通信に付加されるが，これは，実装に用いた XMCAPI がチューニング不足等のため十分に性能が出ていないためである．

今回，評価を 2 ノード上でしかとっていないため，3 ノード以上で評価をし，性能について検証する必要がある．加えて，現状ではチェックポインティング/リスタート機構が 32bit アーキテクチャにしか対応せず，64bit アーキテクチャに対応できていないため，64bit アーキテクチャへの対応を早急に進める予定である．さらに，本実装の完了後，実際に組込み向けネットワークを用いて SCASH/MCAPI の動作を確認する必要がある．

謝　辞

本研究の一部は科学技術振興機構・戦略的創造研究推進事業 (CREST) 研究プロジェクト「実用化を目指した組込みシステム用ディペンダブル・オペレーティングシステム」，研究課題「省電力でディペンダブルな組込み並列システム向け計算プラットフォーム」による．

参 考 文 献

1) Jinpil Lee and Mitsuhisa Sato, Reliable Software Distributed Shared Memory using Page Migration. The Fifteenth International Conference on Parallel and Distributed Systems (ICPADS'09), pp.456-463.
2) Multicore Communications API Working Group: Multicore Communications API, http://www.multicore-association.org/workgroup/mcapi.php.
3) Multicore Association, http://www.multicore-association.org/.
4) Tread Marks, http://www.cs.rice.edu/willy/treadMarks/overview.html.
5) JIAJIA, http://www_users.cs.umn.edu/tianhe/paper/dist.htm.
6) PC Cluster Consortium, http://www.pccluster.org/.
7) Message Passing Interface Forum, http://www.mpi-forum.org/.
8) 小島 好紀ほか, MPI 上のソフトウェア分散共有メモリシステム. 情報処理学会研究報告 (第 98 回 HPC 研究会),2004,pp. 43-48.
9) 三浦信一，鈴木良平，塙敏博，朴泰祐，佐藤三久, 組込み機器向け on-chip/off-chip コア間通信機構の実装と評価. 情報処理学会研究報告，2010-EMB-16(40)，pp. 1-7.