Vol.2010-ARC-190 No.14
2010/8/3

Cellクラスタにおけるスレッド仮想化環境に用いるキャッシュ機構の実装

山田昌弘[†1]　西川由理[†1]
吉見真聡[†2]　天野英晴[†1]

クラスタにおける並列分散処理を行うプログラムは, 一般に MPI 等のノード間通信ライブラリを用いた並列プログラミングにより実装される．しかし, マルチコアプロセッサを用いたクラスタを利用するには，ノード間に加えノード内でのマルチスレッドプログラミングが必要であり，性質の異なる並列プログラミングの知識 · 技術の習得が求められる．

そのため，我々はネットワーク上に接続された複数ノードの演算コアを，仮想的に 1 つのプロセッサ内にあるかのように見せかけ，マルチスレッドプログラミングのみで多数のノードに配置される演算コアを利用出来るスレッド仮想化環境を提案してきた．この環境を用いると，複数ノードの計算資源を容易に活用することができる一方，ノード間の通信遅延が大きく，並列化の効果が得られない問題があった．この問題に対応するため，本研究報告ではノード間通信の遅延を減らす目的で実装されるキャッシュ機構とその評価について述べる．

キャッシュ機構によりノード間通信の回数が削減され，文字列編集距離並列アプリケーションではデータ転送時間を従来の 5%程度に短縮することができた．

The implementation of cache mechanism on the thread virtualization environment for cell cluster

Masahiro Yamada,[†1] Yuri Nishikawa,[†1]
Masato Yoshimi[†2] and Hideharu Amano[†1]

Generally, we use inter-node communication libraries such as MPI for parallel distributed processing in a cluster. However, for a cluster with multi-core, in addition between nodes, it is needed for us to program a multi-thread programming between cores. So, we must learn two types of parallel programming that have different nature.

Therefore, we have proposed the thread virtualization environment which virtualizes us multiple cores in multiple nodes connected to a network as if they are in one node, so only multi-thread programming is required to use many processing cores. If we use this environment, we only need the knowledge of multi-thread programming techniques to effectively utilize the computing resources in multiple nodes. However, long inter-node communication delay could possibly downgrade performance in some applications. In this report, we implemented and evaluate tha cache mechanism for reducing the delay of inter-node communication. The count of inter-node communication is reduced by the cache mechanism, and on Leven Schtein Distance application, we confirmed that the time of inter-node data transfer is shortened as much as 5%.

1. 背　　景

近年の高性能計算 (HPC) 分野のプラットフォームは，汎用の PC 群を相互接続することにより構成される PC クラスタが主流となっている．2005 年頃から，Intel の Quad-Core プロセッサや Core i7(4 コア)，性質の異なる 9 つのコアから成る Cell Broadband Engine(Cell/B.E.) など商用でもマルチコアプロセッサが一般的となり，高い演算能力を示している．このような状況を受けてクラスタの計算ノードは，高周波数と大容量キャッシュによる旧来的な高性能プロセッサから，マルチコアプロセッサを効率よく利用する方向へと変化している．実際，スパコンランキングの世界的な権威である TOP500 プロジェクトによると，2009 年 11 月時点で，4 コア以上のプロセッサを用いたクラスタが，500 台の 90%以上を占めるようになっている[1)]．また，280 個の演算エレメントを持つ GPU を多数接続したクラスタ[2)] や，PowerXCell 8i[3)] のような非対称型プロセッサを用いたクラスタ，GRAPE-DR[4)] のように極めて多数のシンプルな演算コアから成る専用プロセッサから成るクラスタの構築例もある．

これらの専用プロセッサは，汎用 CPU と比べてピーク性能が高く，コスト対性能比や電力性能比なども優れることが多いため，個人や中小規模の組織で構築 · 運営するクラスタのプラットフォームとしての期待が大きい．ところが，これらを利用して高効率計算を実現するには，各プロセッサごとに独自のプログラミングモデルを理解しコーディング上の定石を積み上げる必要があるため，実装が容易であるとは言いがたい．

†1 慶應義塾大学理工学部　〒 223-8522 横浜市港北区日吉 3-14-1
Keio University, 3-14-1 Hiyoshi, Yokohama, 223-8522 Japan
†2 同志社大学理工学部　〒 610-0321　京都府京田辺市多々羅都谷 1-3
Doshisha University, 1-3 Miyakodani, Kyoto, 610-0321 Japan

そのため上記で述べたクラスタで実行されるアプリケーションは，多体問題や Linpack ベンチマークのように，ベクトル型マシンや超並列計算環境での動作実績があるものが主である．データフローが複雑な並列プログラムをこのようなクラスタで実行するには，対象問題を粗粒度な処理と細粒度な処理に切り分け，さらにコア間 · ノード間のデータ通信を明示的に記述する必要があり，高度な技術を要する．

我々は，ネットワーク上に接続された複数ノードの演算コアを，仮想的に 1 つのプロセッサ内にあるかのように見せかけ，多数の演算コアを用いたマルチスレッドプログラミングを行うことのできる，スレッド仮想化環境を提案してきた[5]．この環境により，以下の利点が得られると考えられる．

- プログラマビリティの向上: 主にマルチコアプログラミング技術のみで，ネットワーク上に接続された多数の演算ノードを利用でき，MPI やその他のノード間通信の実装が不要となる
- 柔軟性: ネットワーク上に接続したノード数に応じ，実行に用いるノード数，ノード内のコア数を指定して柔軟にプログラムを実行することができる
- 高互換性: すでに開発されたマルチコアプログラムを，極めて少ない変更点のみで，多ノードで実行可能となる

以上を実現し，動作検証，評価するにあたり，我々の研究グループが構築した Sony 製の PlayStation3(PS3) による Cell クラスタを対象に，ネットワーク上に接続された多数の SPE を利用してプログラムを実行する環境を構築した．しかし，スレッド仮想化環境は socket 通信を用いていたため，ノード間通信による遅延の影響により，データ転送回数の多いアプリケーションにおいては期待される性能向上が得られなかった．

そのノード間通信による遅延の影響を隠蔽するため，各ノードの上にダイレクトマップ方式のキャッシュ機構を設計 · 実装した．各ノードからホストに対してのデータ読み込み要求が発生すると，まず各ノード上に要求されたデータがキャッシュされているかを調べ，キャッシュされていればそのデータを読み込むことにより，ノード間通信回数の削減を試みた．また，評価用のアプリケーションとして文字列編集距離計算アプリケーションを用いて，キャッシュ機構の効果を検証した．

以後，第 2 章では関連研究について第 3 章では Cell/B.E. の基礎知識と Cell クラスタにおける典型的な並列分散処理について, それぞれ述べる．第 4 章ではスレッド仮想化環境の設計と実装について，第 5 章では今回実装したキャッシュ機構の設計と実装について，第 6 章では評価用アプリケーションを用いたキャッシュ機構による性能評価について述べる．最後に，第 7 章で得られた結果を元に結論を述べる．

2. 関 連 研 究

我々が提案しているスレッド仮想化環境はネットワーク上の各ノードを計算資源として活用するものである．関連研究として各ノードをメモリ資源として活用しているものがある．プログラムで大容量メモリを使用したい場合，一台のマシン上にはスロット数などの制限があり，物理的に大容量のメモリを搭載しようとすると非常に高価なものになってしまう．そのため，ネットワーク上のノードのメモリ資源を仮想的に一つの大容量メモリとみなして利用できるような環境（DLM:Distributed Large Memory）[6] が提案されている．これは独自のライブラリを用いたユーザプログラムの最小限の変更と専用のコンパイラによって，ネットワーク上の各メモリ資源を仮想的な大容量メモリとして利用できるようにするものである．

また，処理を複数の実行ユニットに分散して実行する場合に，そのユニット間の通信遅延が大きなボトルネックとなってしまう傾向がある．例えばクラスタ型プロセッサにおいて複数のコアに対して処理を分散させる際，各コアのメモリ参照遅延によって，相対的に性能への影響が大きくなる．そのための改善策として，分散投機メモリフォワーディングなどの研究も行われている[7]．

今回，スレッド仮想化環境においては，ノード間の通信遅延が問題となっている．その問題の改善のため，各ノード上にキャッシュ機構を実装して，遅延の大きいノード間通信の回数を削減させた．

3. Cell Broadband Engine とクラスタ

Cell/B.E. は Sony Computer Entertainment，東芝, IBM によって開発されたマイクロプロセッサである．Cell/B.E. は単純化したインオーダ方式の命令のパイプラインを採用することで回路規模を小さくし，実験室レベルの試作チップは，4GHz を超える高周波数での動作を実現している[8]．

Cell/B.E. の構成を**図 1** 及び**図 2** に示す．Cell/B.E. は，1 つの CPU 内に 9 個のプロセッサをもつ．1 つの PPE および 8 つの SPE からなるヘテロジニアス (非対称) マルチコアプロセッサである．各プロセッサは Element Interconnect Bus (EIB) と呼ばれる高速なバスで接続されている．しかし PS3 の場合，システム制御や製造段階での歩留まりの都合上，ユーザが実際に使用できる SPE の数は 6 つになっている．また，EIB は外部入出力デバイス

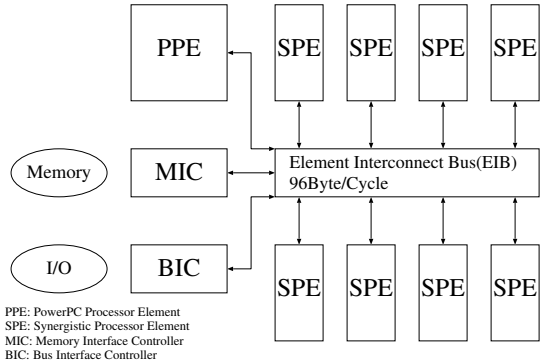

図 1 Cell/B.E. の構成
Fig. 1 The structure of Cell/B.E.

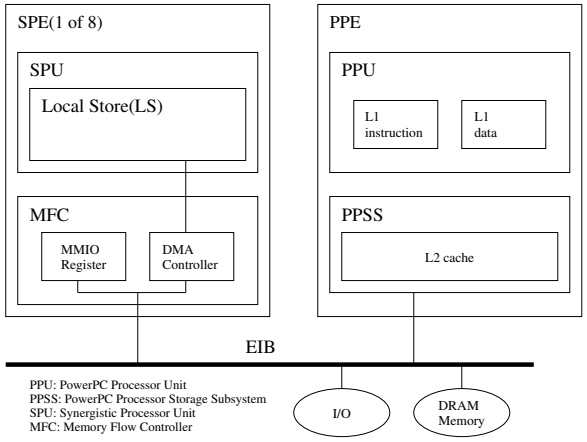

図 2 PPE 及び SPE の構成
Fig. 2 The structure of PPE and SPE.

やメインメモリとも接続されており，各プロセッサ間は EIB を経由してデータ通信を行う．

PPE は，メインメモリや外部デバイス等の制御をおこなう汎用プロセッサである．PPE は PowerPC Processor Unit (PPU) および PowerPC Processor Storage Subsystem (PPSS) から構成され，PPU は命令用およびデータ用にそれぞれ 32KB の一次キャッシュ，PPSS には 512KB の二次キャッシュを備えている．また PPU は PowerPC アーキテクチャをベースとした命令セットを持ち，128 ビット SIMD ユニットである VMX を搭載している．但し，PPE における VMX 命令は倍精度浮動小数点演算には対応していない．PPE は，主に計算のみを行う SPE と比べ汎用性が高い．Cell/B.E. においては，主に SPE への命令実行，及び OS の管理に用いられる．

SPE は Synergistic Processor Unit (SPU)，Local Store (LS)，Memory Flow Controller (MFC) からなる 128 ビット SIMD 型のプロセッサである．SPE は公開されている libspe2 ライブラリを用いて C などの高級言語による実装が可能である．SPU は 128 ビット長のレジスタを搭載した SIMD 命令を持つ演算器である．1 サイクルあたり 4 並列で演算を行うことが可能であるが，SPE における単精度浮動小数点演算において丸め誤差は切り捨てられ，IEEE754 に準拠しない．SPE は LS は 128bit/cycle アクセスが可能な 256KB の専用メモリを持っている．

現在の PPE×1+SPE×8 という実装の下では，EIB は 16Byte/CPU Cycle×4 本のリング構造からなり，各プロセッサ及び MIC，BIC を接続し，データの転送に用いられる．各バスでは 3 つの DMA 転送が行われるため，EIB は 96Byte/Bus Cycle の帯域をもつ．同一リング状を同時に複数のトラフィックが走る，通常のトークンリングとは異なる特殊な構造をもつ．EIB は大きな帯域を持ったバスであるが，PPE·SPE は各々16Byte/CPU Cycle であるため十分とは言いがたく，帯域を考慮したデータ通信を行う必要がある．

4. スレッド仮想化環境

通常の Cell クラスタの並列分散処理プログラミングの概念を**図 3** に示す．

計算ノードに PS3 を用いた Cell クラスタにおいては，各ノード間の通信制御は MPI を用いて行い，各ノードが持つ 6 個の SPE で計算を行う．しかし，これには MPI プログラミングと Cell プログラミングの両方を組みあわせたプログラムコードを書かねばならず，この方法は一般的にデバッグが難しく，高度なプログラミング技術が必要となる．そこで，本研究ではホストとなるマシンの PPE が，全てのノードの SPE をまとめて利用できるミドルウェアを開発した．これにより，プログラマは MPI プログラミングを意識せず，使用できる SPE の数を（ノード数）×（SPE 数）まで拡張した並列分散処理プログラムを書くことができる．

4.1 スレッド仮想化環境の構造

スレッド仮想化環境のイメージを**図 4** に示す．

また，これを実現するための構造を**図 5**，および**表 1** に示す．スレッド仮想化環境では，

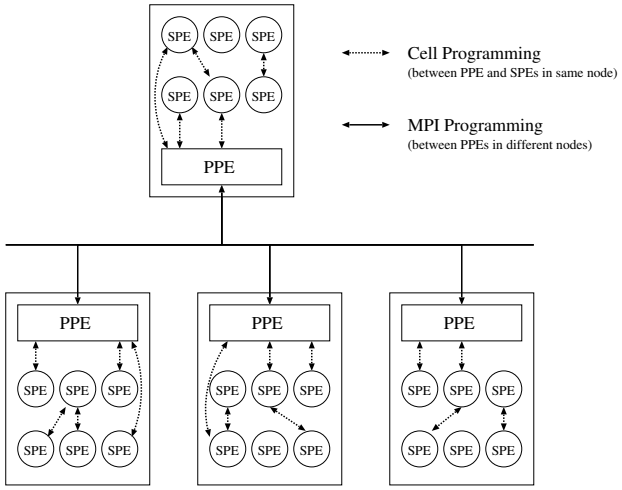

図 3 Cell クラスタの並列分散処理
Fig. 3 A parallel distributed process of the Cell cluster.

表 1 設計した各サーバ ·API の役割
Table 1 Roles of each server and API we designed.

仮想 SPE API	仮想 SPE Server へと SPE 制御要求を出す
仮想 SPE Server	各ノードの SPE を制御
仮想 DMA API	仮想 DMA Server へと DMA 転送要求を出す
仮想 DMA Server	ホストとノード間のデータ転送を仲介

ホスト上に仮想 DMA 転送サーバ，各ノード上に仮想 SPE 制御サーバと仮想 DMA 転送サーバが常駐している．各ノードの SPE とホストは，仮想通信用の API を用いてサーバ群にリクエストを送信し，通信を仲介する．

一般的に，PPE と SPE 間の通信には主に 2 種類ある．

(1) SPE 制御用通信

(2) DMA 転送

これらの通信を実現する関数群はそれぞれ libspe2．h ライブラリと spu_mfcio．h ライブラリ内で実装されている．しかし，これらの関数は同じマシン内の PPE·SPE 間通信のためとなっており，他ノードとの通信には対応していない．

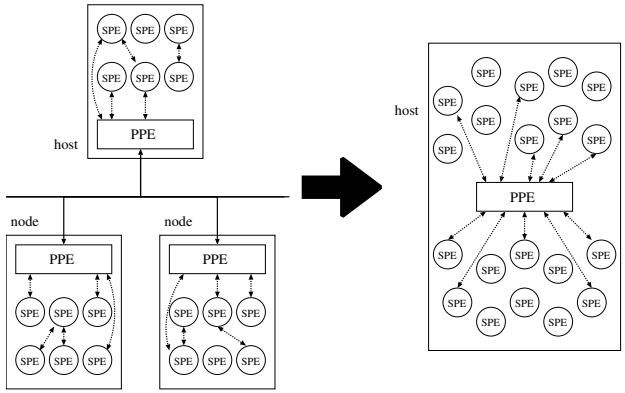

図 4 スレッド仮想化環境のイメージ
Fig. 4 Image of the Thread Virtualization Environment.

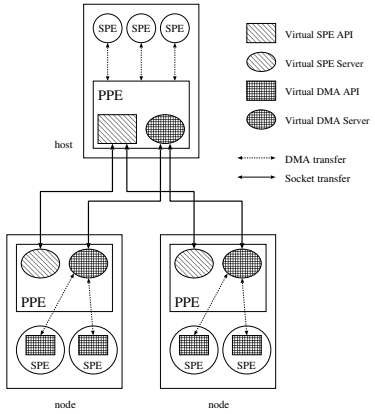

図 5 設計したスレッド仮想化環境の構造
Fig. 5 The structure of the Thread Virtualization Environment we designed.

上記 2 つの通信をスレッド仮想環境へ拡張するために以下のような設計を行った．

4.1.1 仮想 SPE 制御

他ノードの SPE をホストから制御するために，各ノードの PPE 上に SPE 制御サーバを常駐させ，ホストの PPE 上から仮想 SPE 制御 API を用いてアクセスする．仮想 SPE 制

御用 API として**表 2** 関数を実装した．

これらは全て，内部では各ノードの SPE 制御サーバへとリクエストを送信し，各ノードの SPE 制御サーバ上で実際の処理が行われる．

4.1.2 仮想 DMA 転送

ホストと他ノードの SPE の間でのデータ転送を実現するために，仮想 DMA 転送用 API として以下の関数を作成した．

- v_spu_mfcdma64()･･･ ホストのメインメモリとデータ転送を行う

この関数は，自身の PPE 上で動作している仮想 DMA 転送サーバに対してリクエストを送信し，さらにその仮想 DMA 転送サーバがホスト上の仮想 DMA 転送サーバと socket 通信を行うことにより，各ノード - ホスト間のデータ伝送を実現する．ホストのメインメモリから各ノードの SPE までデータを読み込む GET 転送には，この関数の第 6 引数に，MFC_CMD_GET を指定する．また，各ノードの SPE からホストのメインメモリへとデータを書き込む PUT 転送には，MFC_CMD_PUT を指定する．今回これら 2 つの引数に加え，各ノード上存在するキャッシュテーブルを利用して GET 転送をすることが可能になる MFC_CMD_GET_CACHE という引数を用意した．

詳細については次章で述べる．

5. キャッシュ機構

本研究で導入したキャッシュ機構の設計と実装について述べる．キャッシュ機構は，ノード間通信の回数を減らして通信遅延を抑えるため，各ノードの SPE がホストマシンのメインメモリからデータを転送（GET）する際に，同一ノード上の PPE のメインメモリにデータをキャッシュする機能を持つ．

5.1 設　　計

各ノードの PPE 上にダイレクトマップ方式のキャッシュ機構として機能するプログラムを設計した．キャッシュ機構の仕組みを**図 6** に示す．

表 2　仮想 SPE 制御用 API
Table 2　API for virtual SPE control.

v_spe_image_open()	ホストと全ノード上に SPE プログラムファイルをオープン
v_spe_context_create()	ホスト上かノード上に SPE コンテキストを生成
v_spe_program_load()	ホスト上かノード上に SPE プログラムをロード
v_spe_context_run()	ホスト上かノード上の SPE プログラムを実行開始
v_spe_context_destroy()	ホスト上かノード上の SPE コンテキストを破棄
v_spe_image_close()	ホストと全ノード上の SPE プログラムをクローズ

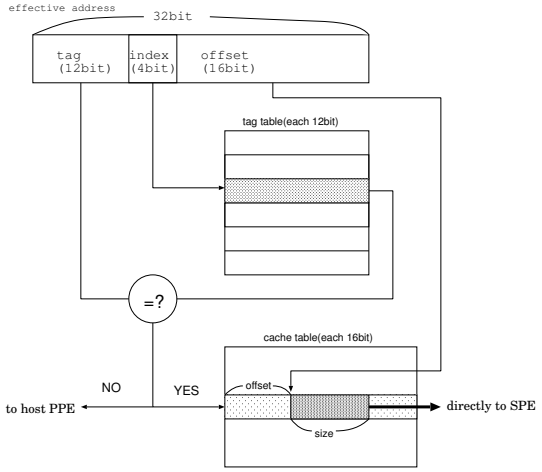

図 6　キャッシュ機構の仕組み
Fig. 6　The cache mechanism.

各ノードの PPE 上の仮想 DMA サーバは，SPE からの GET 要求を受け付けると，要求されたデータの実効アドレス (EA) をタグ部，インデックス部，オフセット部に分け，タグとインデックスを元にキャッシュされているかを調べる．キャッシュされていた場合はノード間通信をすることなく SPE に要求されたデータを転送することができる．キャッシュされていなかった場合はホスト PPE とのオフセットサイズ分のデータ転送が発生し，新たにデータがキャッシュされてタグテーブルが更新される．

また，図 6 ではタグサイズが 12bit，インデックスサイズが 4bit，オフセットが 16bit となっているが，これらはプログラマがアプリケーションに合わせて，変更可能である．

5.2 実　　装

既存の spu_mfcdma64 関数の第 6 引数が DMA 転送の挙動を指定する．引数として通常選択されるのは以下の 2 種類である．

- MFC_GET_CMD
- MFC_PUT_CMD

本研究では，これに加えて以下のコマンドを引数として使用できるように実装した．

- MFC_GET_CMD_CACHE

このコマンドを使用することで，キャッシュ機構を利用した GET データ転送を行うことができる．

また各ノードのメインメモリ上に以下の 2 種類のテーブルを実装した．

- タグテーブル
- キャッシュテーブル

タグテーブルは要求されたキャッシュブロックが，キャッシュテーブルに格納されているかを知るためのタグを管理する．キャッシュテーブルには実際のキャッシュブロックが格納されている．デフォルトではインデックス · キャッシュテーブルのエントリ数は 16(= 2^4) で，キャッシュブロックサイズは 64KB だが，ブロックサイズもユーザが変更できる．

現在では，ホストマシンと各ノード上のキャッシュ間での同期機構は実装していない．よって，プログラマが自身の判断で同期の必要のない領域において，キャッシュを利用したデータ転送を MFC_GET_CMD_CACHE 命令を使用する．

6. 評　　価

この章では, スレッド仮想化環境におけるキャッシュ機構の性能評価について述べる．ベンチマークアプリケーションとして文字列編集距離の計算プログラム[9] を使用した．

今回の評価は**表 3** の環境で行った．ホストマシンとして BCU-100，ノードマシンとして PlayStation3 を用いた．

6.1 文字列編集距離計算

設計と実装を行ったスレッド仮想化環境のキャッシュ機構の性能評価について述べる．

文字列の編集距離（LevenSchteinDistance）は 2 つの文字列の距離を数値化したものであり，かな漢字変換エンジンや, DNA の相同性検索などに利用されている．2 つの文字列の距離は, 片方の文字列を削除 · 挿入 · 置換のいずれかの操作でもう一方の文字列を得るための操作回数の最小値で定められる．

この計算は**図 7** のように，マトリックス上の左上から右下にかけて処理が進んでいく．各ブロックを計算するためには，そのブロックの上 · 左上 · 左のブロックの計算結果が必要となる．そのため処理の特徴として，最初は並列度は低いものの，処理が進むに連れて並列度が高くなることが挙げられる．そして, 終盤に近づくにつれ並列度が低くなっていく．また, ノード間 · コア間のデータの受け渡しなどでデータ転送を頻繁に行っているのも特徴である．編集距離を Cell/B.E. のようなマルチコアプロセッサで計算する場合，単純にデータを等分割して各コアに演算させるのではなく，ホストがジョブキューを管理し，各コアに逐次計算ブロックを割り振るアルゴリズムが効率的と考えられる．この実装では，データ転送が頻繁に行われている粒度の細かいアルゴリズムなので，スレッド仮想化環境では，キャッシュを用いない場合 socket 通信による遅延の影響が極めて大きく，複数ノードを使う効果が得られていなかった[5]．

表 3　評価環境
Table 3　An environment of evaluation.

ホスト	BCU-100
ノード	PlayStation3
CPU	Cell Broadband Engine 3.2GHz
Memory	256MB(PS3), 1GB(BCU-100)
OS	Yellow Dog Linux 6.0
PPE コンパイラ	ppu-gcc 4.1.1
SPE コンパイラ	spu-gcc 4.1.1
ノード間接続	GbE

SPE の GET データ転送において，一部を通常の MFC_CMD_GET を用いてデータ転送を行った場合と，MFC_CMD_GET_CACHE を用いてキャッシュを利用してデータ転送を行った場合との実行時間の比較を行った．キャッシュサイズも 64KB（= 2^{16}）と 256KB（= 2^{18}）の 2 種類で測定した．

6.2 評 価 結 果

図 8 に SPE 数を横軸，GET データ転送の総実行時間を縦軸にとったグラフを示す．

結果は，キャッシュを使用しない場合と比較して，キャッシュサイズ 64KB のときは GET データ転送の実行時間が増加したが，256KB のときは効果が見られた．特に SPE を 12 個使用したときには実行時間をおよそ 5%に短縮できた．

キャッシュサイズ 64KB のときに性能が悪化した理由としては，キャッシュサイズの大きさが十分ではなく，頻繁にキャッシュテーブルの更新が発生してしまったためと考えられる．また，キャッシュサイズが 256KB のときに性能が大きく改善されたのは，このアプリケーションに対してキャッシュサイズが十分に大きく，キャッシュテーブルがほぼ更新されなかったため，メインメモリからのデータ転送回数を削減できたためと言える．

7. 結　　論

本研究報告ではスレッド仮想化環境における大きな問題点の 1 つであるノード間通信の遅延について，各ノードの PPE 上にキャッシュ機構を実装することにより，遅延の削減を

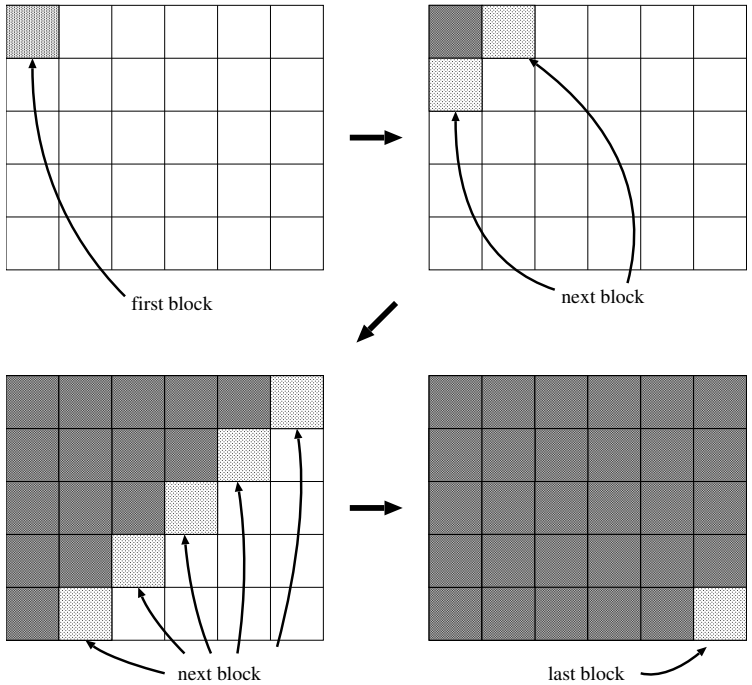

図 7 文字列編集距離計算の並列度の特徴
Fig. 7 The feature of parallelism about Leven Shtein Distance.

図った．またその効果を検証するため，文字列編集距離アプリケーションを用いて評価を行った．

その結果，キャッシュ機構の有効性と適切なキャッシュサイズ選択の必要性の 2 点が確認された．

キャッシュ機構を用いることによって，各ノードの SPE の GET データ転送の通信時間が 5%程度に短縮されたが，適切なキャッシュサイズを選択しないと，キャッシュテーブルの更新が頻繁に発生し，期待する性能向上を得ることができない．

現在，各ノードのキャッシュテーブル間の同期機構は実装していない．よって，現時点ではテーブル間で同期をとる必要がないデータ転送のみキャッシュ機構を利用できる．よって本仮想化環境の対象アプリケーションが限定される．そのため今後の課題としては，様々な

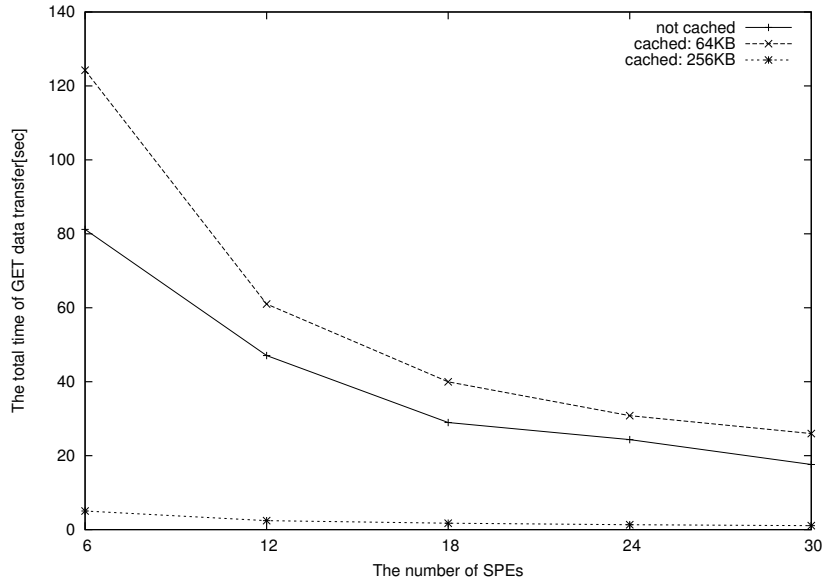

図 8 GET データ転送時間の比較
Fig. 8 The comparison between non-cache and cached data transfer.

アプリケーションに対してもキャッシュ機構を利用できるようにするため，各ノードのキャッシュ間の同期機構の実装を行う予定である．

参 考 文 献

1) Top500: ”Supercomputer sites”. ”http://www.top500.org/system/10377”.
2) Hamada, T., Narumi, T., Yokota, R., Yasuoka, K., Nitadori, K. and Taiji, M.: 42 TFlops hiearchcal N-body simulations on GPUs with applications in both astrophysics and turbulence, *SC’09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pp.1–12 (2009).
3) Kistler, M., Gunnels, J., Brokenshire, D. and Benton, B.: Programming the Linpack benchmark for the IBM PowerXCell 8i processor, *Scientific Programming*, Vol.17, No.1-2, pp.43–57 (2009).
4) Makino, J., Hiraki, K. and Inaba, M.: GRAPE-DR: 2-Pflops massively-parallel computer with 512-core, 512-Gflops processor chips for scientific computing, *SC’07: Proceedings of the Conference on High Performance Computing Networking, Stor-*

age and Analysis (2007).
5) 山田昌弘，西川由理，吉見真聡，天野英晴：Cell Broadband Engine を用いたスレッド仮想化環境の提案，信学技報， Vol.110, No.2, pp.27–32 (2010).
6) 緑川博子，黒川原佳，姫野龍太郎：遠隔メモリを利用する大容量メモリシステム DLM とコンパイラ，情報処理学会ハイパフォーマンスコンピューティング研究会資料，Vol.115, No.7, pp.37–42 (2008).
7) 入江英嗣，服部直也，高田正法，坂井修一，田中英彦：クラスタ型プロセッサのための分散投機メモリフォワーディング，*SACSIS2004*, Vol.2004, No.6, pp.177–186 (2004).
8) 近藤伸宏：”CELL プロセッサに見るアーキテクチャ - 次世代デジタルホームに向けて”，東芝レビュー， Vol.60, No.7, pp.48–51 (2005).
9) マルチコアプログラミングコンテスト：”Cell Challenge 2009 ツールキット解説書 ver.1.0 対応版”. ”`http://www.hpcc.jp/sacsis/2009/cell/`”.