DS-CUDA ソフトウェアパッケージ ユーザガイド

for DS-CUDA version 1.2.2

最終更新：2012 **年** 9 **月** 6 **日**

川井 敦
E-mail: kawai@kfcr.jp

目 次

1 本文書の概要 **3**

2 DS-CUDA 概観 **3**
- 2.1 **機能概要** . . . 3
- 2.2 **システムの構成** . . . 4
- 2.3 **ソフトウェア階層** . . . 4
- 2.4 **クライアントプログラムの生成** . . . 5
- 2.5 **冗長デバイス** . . . 5

3 インストール **6**
- 3.1 **準備** . . . 6
- 3.2 **パッケージの展開** . . . 7
- 3.3 **環境変数の設定** . . . 8
- 3.4 **ライブラリ・実行ファイルのビルド** . . . 9
- 3.5 **動作チェック** . . . 9
  - 3.5.1 **サンプルプログラム** . . . 9
  - 3.5.2 CUDA SDK **サンプルプログラム (準備中)** . . . 9

4 使用方法 **10**
- 4.1 **アプリケーションプログラムのビルド** . . . 10
- 4.2 **アプリケーションプログラムの実行** . . . 11
  - 4.2.1 Sever node **の設定** . . . 12
  - 4.2.2 Client node **の設定** . . . 13
- 4.3 **冗長デバイスによる誤り検出と自動再計算** . . . 13
  - 4.3.1 **エラーハンドラの設定** . . . 14
  - 4.3.2 **自動再計算** . . . 14

5 DS-CUDA 実装の詳細 **16**
- 5.1 CUDA **ランタイムライブラリのサポート範囲** . . . 16
- 5.2 CUDA C/C++ **文法のサポート範囲** . . . 16
- 5.3 InfiniBand Verb **インタフェース** . . . 17
- 5.4 RPC **インタフェース** . . . 17
- 5.5 Client node **から** server node **への** CUDA **カーネルの転送** . . . 17

6 利用許諾 **18**

7 DS-CUDA ソフトウェアパッケージ更新履歴 **18**

1 本文書の概要

この文書では DS-CUDA ソフトウェアパッケージの使用方法を説明します。DS-CUDA は PC の I/O スロットに接続された NVIDIA 社製 GPU カード (CUDA デバイス) を、ネットワーク接続された他の PC からシームレスに使用するためのミドルウェアです。本バージョンは CUDA バージョン 4.1 で動作確認ずみですが、CUDA バージョン 4.1 の提供するすべての機能をサポートしているわけではありません (節 5.1 参照)。

以降、第 2 章では DS-CUDA の基本構成と動作概要を説明します。第 3 章では DS-CUDA ソフトウェアパッケージ (以降「本パッケージ」と呼びます) のインストール方法を説明します。第 4 章ではアプリケーションプログラムのコンパイル方法、実行方法について説明します。第 5 章では DS-CUDA の実装や内部動作について触れます。

なお以降では、本パッケージのルートディレクトリ (/パッケージを展開したディレクトリ/dscudapkg バージョン番号/) を $dscudapkg と表記します。

2 DS-CUDA 概観

本節では DS-CUDA の基本構成と機能を概観します。

2.1 機能概要

DS-CUDA を用いると、アプリケーションプログラムはネットワーク上に分散配置された GPU を透過的に扱えます。例えば下図の Client Node 上のアプリケーションは、Client Node 自身にローカルにインストールされた GPU へアクセスする場合と同じプログラミングインタフェースを用いて Server Node に接続された GPU へアクセスできます。

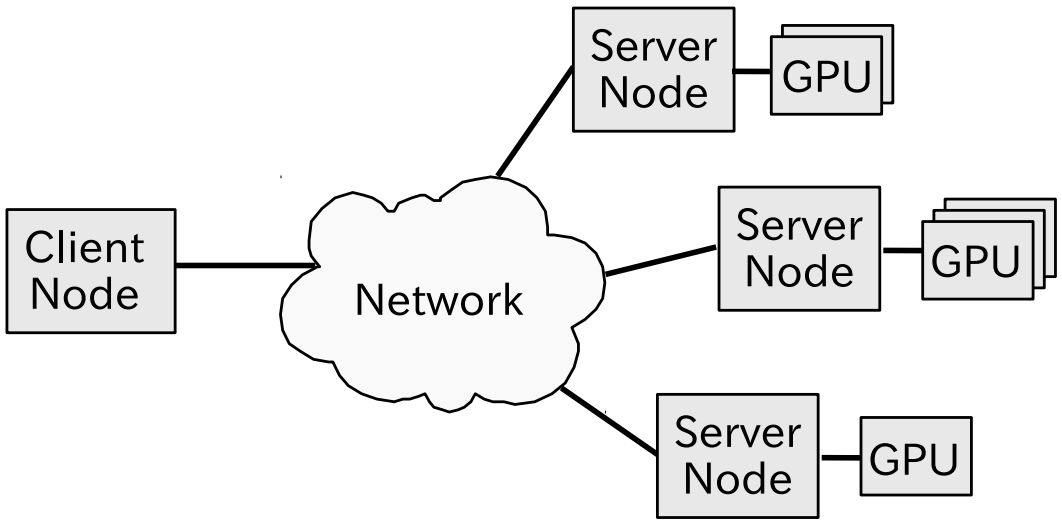
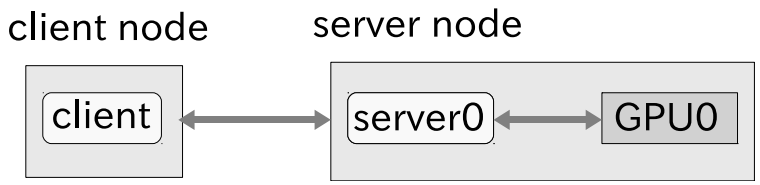


図 : GPU クラスタの例。

2.2 システムの構成

もっとも単純な例として、PC 2 台と GPU 1 台からなるシステムを考えます。



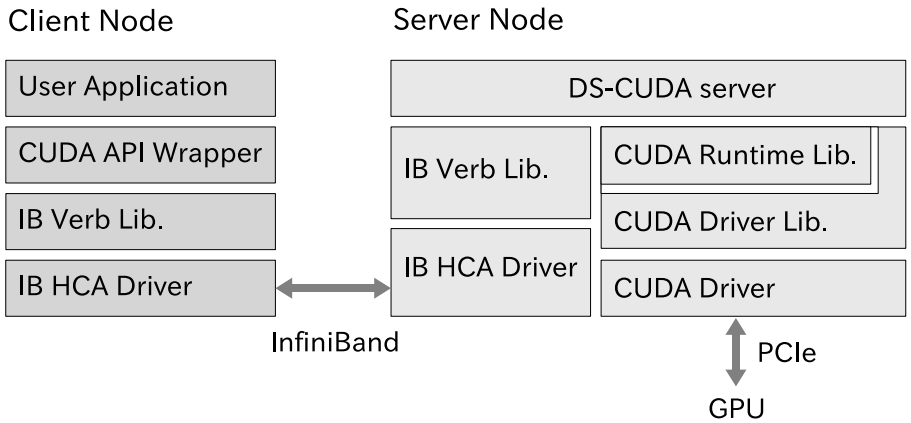
図：最小構成の DS-CUDA システム。

このシステムは互いにネットワーク接続された 2 台の PC と、その一方にインストールされた GPU から構成されます。GPU は NVIDIA 社の CUDA に対応した製品 (CUDA デバイス) であることが必須です。ネットワーク接続には原則として InfiniBand の使用を想定します。ただし TCP/IP による通信が可能なネットワークであれば、InfiniBand 以外のものも使用できます (例: GigabitEthernet)。簡便のため、GPU を持つ側の PC を server node と呼び、もう一方の PC を client node と呼ぶことにします。

DS-CUDA はユーザに対し、クライアント・サーバ型の実行環境を提供します。serve node 上ではサーバプログラム `dscudasvr` を常時稼働させておき (図中の server0)、client node 上のアプリケーションプログラム (図中の client) を `dscudasvr` に対するクライアントとして実行します。サーバプログラムはアプリケーションプログラムの要求に従って GPU を制御します。

2.3 ソフトウェア階層

DS-CUDA のソフトウェアは下図に示すように階層化されています。



図：ソフトウェア階層

client node 上のユーザアプリケーションは CUDA API を用いて GPU (CUDA デバイス) にアクセスします。内部的にはこれは server node 上のサーバプログラムへのアクセスに置き換えられますが、ユーザアプリケーションはそのことを意識する必要はありません。クライアント・サーバ間の通信プロトコルには InfiniBand Verb もしくは TCP/IP を使用します (図のソフトウェア階層は InfiniBand Verb を使用した場合のもの)。

2.4 クライアントプログラムの生成

CUDA C/C++ で記述したユーザアプリケーションのソースコードを、本パッケージの提供する DS-CUDA プリプロセッサ `dscudacpp` を用いてコンパイルすることにより、client node で動作するプログラムを生成します。

2.5 冗長デバイス

DS-CUDA は冗長デバイスを扱えます。冗長デバイスとは複数の GPU を用いて構成された単一の仮想デバイスです。この仮想デバイス上で計算を実行すると、仮想デバイスを構成する複数の GPU 上で同一の計算が行われ、両者の結果が異なっていた場合には、その旨がユーザアプリケーションに通知されます。

アプリケーションプログラムが一定の制約 (節 4.3 参照) を満たすように記述されている場合には、誤りを検出した計算を自動的に再実行させることも可能です。

3 インストール

3.1 準備

本パッケージは以下のソフトウェアに依存しています。インストール作業の前に、これらの動作環境を整えて下さい。

- CUDA **開発ツール** (CUDA 4.1 **で動作確認済**)
  `http://www.nvidia.com/`
- C++ **コンパイラ** (g++ version 4.1.0 **以降を推奨**)
  `http://gcc.gnu.org/`
- Ruby (version 1.8.5 **以降を推奨**)
  `http://www.ruby-lang.org/`
- OFED (version 1.5.4 **以降を推奨**)
  `https://www.openfabrics.org/resources/\`
  `ofed-for-linux-ofed-for-windows/linux-sources.html`

注意 :

- **コンパイル対象とするアプリケーションプログラムが** CUDA **カーネルを** C++ **テンプレートとして実装している場合には、**C++ **コンパイラには** g++ version 4.0.0 **以上を使用してください。それ以前のバージョンや、**Intel C++ **コンパイラ等では動作しません。これは** C++ **テンプレートからシンボル名を生成する際の** name mangling **規則がコンパイラごとに異なっており、**DS-CUDA **では現在のところ** g++ version 4 **系の** name mangling **規則のみをサポートしているためです。**
- ruby **は** `/usr/bin/` **にインストールして下さい。他のディレクトリにインストールされている場合には** `/usr/bin/` **へシンボリックリンクを張って下さい。**
- `/etc/security/limits.conf` **に以下の** 2 **行を追記して下さい。**

  ```
  * hard memlock unlimited
  * soft memlock unlimited
  ```

  この設定は root **権限を持たない一般ユーザが** InfiniBand Verb **経由の通信を行うために必要です。**

3.2 パッケージの展開

ソフトウェアパッケージ `dscudapkg`n`.tar.gz` **を展開してください** (n **はバージョン番号**)。**パッケージには以下のファイルが含まれています**:

doc/	**本文書、その他のドキュメント。**
scripts/	**パッケージ管理ユーティリティ。**
bin/	
dscudacpp	.cu **ファイルから** DS-CUDA **クライアントを生成する プリプロセッサ。**
dscudasvr	DS-CUDA **サーバ。**
ptx2symbol	CUDA **カーネルの** name mangling **されたシンボル名を取得 するスクリプト。**`libdscuda.a` **が使用します。**
include/	**ヘッダファイル** (DS-CUDA **クライアント・サーバ共用**)。
lib/	
libdscuda_ibv.a	DS-CUDA **ライブラリ** (**通信プロトコルに** InfiniBand Verb **を使用**)。
libdscuda_rpc.a	DS-CUDA **ライブラリ** (**通信プロトコルに** TCP/IP **を使用**)。
src/	DS-CUDA **サーバ、ライブラリのソースコード。**
misc/	**サーバ構成指定ファイルの例、**make **ファイルのサンプル等。**
sample/	**アプリケーションプログラムの例。**

3.3 環境変数の設定

以下の環境変数を設定してください。

client node, server node **共通**	
CUDAPATH :	CUDA Toolkit **のインストールされているパス。デフォルト値は** `/usr/local/cuda`
CUDASDKPATH :	CUDA SDK **のインストールされているパス。デフォルト値は** `/usr/local/cuda/NVIDIA_GPU_Computing_SDK`
DSCUDA_PATH :	DS-CUDA **ソフトウェアパッケージのインストールされているパス。設定必須。デフォルト値はありません。**
DSCUDA_WARNLEVEL :	DS-CUDA **サーバおよびクライアント実行時のメッセージ出力レベル。整数値を指定します。値が大きいほど詳細なメッセージが出力されます。デフォルト値は** 2**、最小値は** 0 **です。**
server node **のみ**	
DSCUDA_REMOTECALL	**通信プロトコルを選択します。指定できる値は** `ibv`, `rpc` **のいずれかです。それぞれ** InfiniBand Verb, Remote Procedure Call **を意味します。**
client node **のみ**	
LD_LIBRARY_PATH :	**共有ライブラリパスに** `$DSCUDA_PATH/lib` **を追加してください。設定必須。**
DSCUDA_SERVER :	DS-CUDA **サーバが動作している** PC **の** IP **アドレス、あるいはホスト名。デフォルト値は** `localhost` **複数のサーバを使用する場合の記法については節** 4.2.2 **を参照して下さい。**
DSCUDA_SERVER_CONF :	DS-CUDA **サーバが動作している** PC **の** IP **アドレス、あるいはホスト名を記述したファイルのファイル名。環境変数** DSCUDA_SERVER **への設定値が長く煩雑になってしまう場合 (多数のサーバを使用する場合など) 、設定値をファイルに記述し、そのファイル名をこの環境変数で指定できます。**
DSCUDA_AUTOVERB :	**冗長デバイス上で自動再計算機能を使用する場合にこの変数を定義します。変数にはどのような値を設定しても構いません。**

例:

```
kawai@localhost>export DSCUDA_PATH="/home/kawai/src/dscudapkg1.2.2"
kawai@localhost>export DSCUDA_SERVER="192.168.10.101"
kawai@localhost>export LD_LIBRARY_PATH=/home/kawai/src/dscudapkg1.2.2/lib:\
$LD_LIBRARY_PATH
```

CUDA や C コンパイラが参照する環境変数がある場合には、必要に応じてそれらも設定して下さい。

3.4 ライブラリ・実行ファイルのビルド

ディレクトリ $dscudapkg/src へ移動し、make を実行してください。DS-CUDA ライブラリ $dscudapkg/lib/libdscuda_ibv.a、$dscudapkg/lib/libdscuda_rpc.a と DS-CUDA サーバ $dscudapkg/bin/dscudasvr が生成されます。

3.5 動作チェック

$dscudapkg/sample/ 内のサンプルプログラムを使用して、本パッケージの動作を確認します。

3.5.1 サンプルプログラム

$dscudapkg/sample/ 内に各種のサンプルプログラムが格納されています。

- vecadd: ベクトルの加算を行います。
- direct: 重力多体シミュレーションを行います (make run で初期条件を生成し、シミュレーション実行します)。
- claret: 溶融塩のシミュレーションを行います。

各ディレクトリ内で make を実行すると、それぞれの DS-CUDA クライアントが生成されます。

3.5.2 CUDA SDK サンプルプログラム (準備中)

4 使用方法

4.1 アプリケーションプログラムのビルド

CUDA C/C++ で記述されたユーザアプリケーションのソースコード (以下 .cu ファイルと表記します) から DS-CUDA クライアントを生成するには、DS-CUDA プリプロセッサ $dscudapkg/bin/dscudacpp を使用します。
dscudacpp を引数を与えずに実行すると、使用方法が表示されます。

```
kawai@localhost>pwd
/home/kawai/src/dscuda1.2.2/sample/direct
kawai@localhost>../../bin/dscudacpp
No input file given.
  usage: ../../bin/dscudacpp [options] inputfile(s)...
  options:
      --infile <file>   : a .cu input file.
      -i <file>

      -o <file>         : an output file.

      --verbose[=level] : be verbose. the level can optionally be given. [2]
      -v[level]           the higher level gives the more verbose messages. \
  level 0 for silence.

      --help            : print this help.
      -h
  Note that all options not listed above are implicitly passed on to nvcc.
```

dscudacpp へ .cu ファイルを与えるには、オプションスイッチ -i を使用します。また、生成される DS-CUDA クライアントは、オプションスイッチ -o で指定します。その他の入力ファイル (.o や .c など) はオプションスイッチ無しにファイル名だけを引数として与えます。これら以外のすべての引数やオプションスイッチは、dscudacpp では解釈されずに、dscudacpp が内部的に起動する nvcc へと渡されます。nvcc が必要とする引数は、すべて dscudacpp への引数として渡さねばなりません。また以下の引数を与えて DS-CUDA ライブラリをリンクする必要があります。

- 通信プロトコルとして InfiniBand Verb を用いる場合:
  -ldscuda_ibv -libverbs -lrdmacm -lpthread
- 通信プロトコルとして TCP/IP を用いる場合:
  -ldscuda_rpc

例えば、本パッケージ付属のサンプルプログラム $dscudapkg/sample/vecadd/ のク

ライアント userapp を生成するには、下記の引数が必要です (通信プロトコルとして InfiniBand Verb を用いる場合)。

```
kawai@localhost>pwd
/home/kawai/src/dscuda1.2.2/sample/vecadd
kawai@localhost>../../bin/dscudacpp -o userapp -I. -i userapp.cu \
-ldscuda\_ibv -libverbs -lrdmacm -lpthread
```

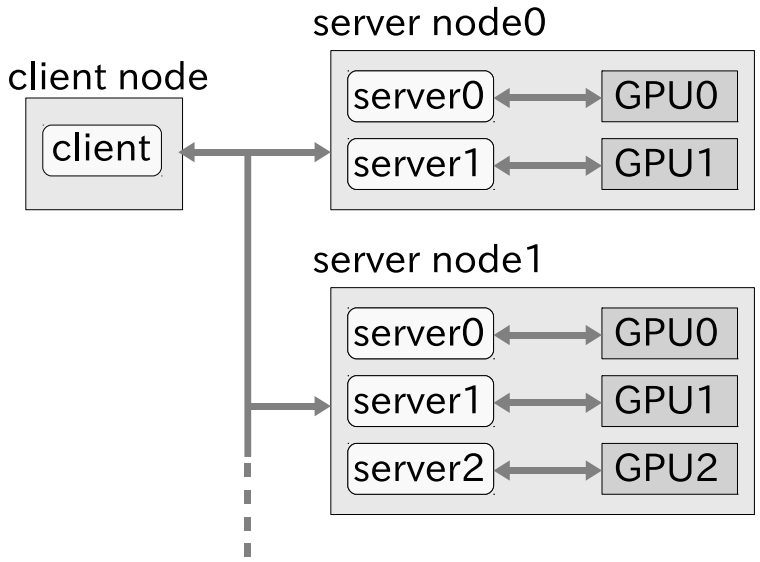
参考 : dscudacpp は以下の C プリプロセッサマクロ定数を定義します。定数はソースコード中で参照できます。

定数名	値
__DSCUDA__	1
__DSCUDACPP_VERSION__	バージョン番号 (例 : バージョン 1.2.3 の値は 0x010203)

4.2 アプリケーションプログラムの実行

client node 上でアプリケーションプログラムを実行するには、server node 上で事前にサーバプログラムを起動しておく必要があります。

またこれまでは簡単のために 1 台の GPU がインストールされた 1 台の server node のみを考えてきましたが、複数の server node を持つシステム (例えば下図) の場合には、client node のアプリケーションプログラムの使用する GPU を client node 側で指定する必要があります。以下ではこれらの設定方法について説明します。



図：**複数の** GPU **を持つシステムの例。**

4.2.1 Sever node **の設定**

client node **が使用するすべての** server node **上で、**GPU **を制御するためのサーバプログラム (より正確にはプロセス) を起動します。**1 **台の** GPU **につき** 1 **つのサーバプログラムが必要です。**

サーバプログラムの実行ファイルは `$dscudapkg/bin/dscudasvr` **です。これをコマンドラインから実行します。**1 **台の** server node **に複数の** GPU **がインストールされている場合には、実行時にサーバプログラム自身の** ID **と** GPU **のデバイス** ID **を次のように指定してください**:

`dscudasvr -s` *server_id* `-d` *device_id*

ここで *server_id* **はサーバプログラムの** ID (**サーバ** ID) **です。この** ID **は** client node **上のアプリケーションプログラムが通信相手を特定するために使用します。各** server node **内でユニークな、小さな非負整数 (特別な理由のない限り** 0, 1, 2, ...) **を割り当ててください。** *device_id* **は** `dscudasvr` **が制御する** GPU **を特定する際に使用する** GPU **のデバイス** ID **です。この** ID **は** NVIDIA **のドライバによって各** GPU **に自動的に割り振られます。**1 **台の** server node **内に** n **個の** GPU **がインストールされている場合、各** GPU **には** 0 **から** $n-1$ **までのいずれかの値がユニークに割り当てられます。割り当て状況を確認するには、例えば** CUDA SDK **に含まれる** `deviceQuery` **等を使用して下さい。**

4.2.2 Client node の設定

client node 側は、アプリケーションプログラムがどの server node 上のどのサーバプログラムと通信するかを、環境変数 DSCUDA_SERVER によって指定します。DSCUDA_SERVER には以下の書式に従って記述した文字列を設定します。

$server\_node$:$server\_id$

ここで $server\_node$ は server node の IP アドレスあるいはドメイン名です。$server\_id$ はサーバプログラムの ID (節 4.2.1 参照) です。次の例のように複数のサーバプログラムを空白で区切って列挙すると、

```
192.168.0.100:0 192.168.0.100:1 192.168.0.105:0 192.168.0.107:1
```

アプリケーションプログラムからは複数の GPU があるように見えます。アプリケーションプログラムから見える仮想的な GPU (以降仮想デバイスと呼びます) のデバイス ID は、列挙した順に 0, 1, 2, ... が割り振られます。

冗長デバイス (節 2.5 参照) を構成するには、複数のサーバプログラムを空白ではなくカンマで区切って列挙します。例えば

```
192.168.0.100:0,192.168.0.100:1,192.168.0.105:0
```

は、server node 192.168.0.100 にインストールされた 2 枚の GPU と 192.168.0.105 にインストールされた 1 枚の GPU、合計 3 枚の GPU を用いて 1 台の冗長デバイスを構成します。

空白区切りとカンマ区切りを混在することも可能です。例えば

```
mysvr0:0 mysvr0:1,mysvr1:0 mysvr2:0
```

は、合計 4 台の GPU を用いて、通常の仮想デバイス 2 台と冗長デバイス 1 台、合計 3 台の仮想デバイスを構成します。

4.3 冗長デバイスによる誤り検出と自動再計算

冗長デバイス (節 2.5、節 4.2.2) を用いて計算を行うと、冗長デバイスを構成する複数の GPU 上で同一の計算が実行され、それらの結果が一致するかどうかが検証されます。一致しなかった場合には、いずれかの GPU で行われた計算の結果が誤っていた、と見なされ、アプリケーションプログラムはエラーメッセージを出力して終了します。

4.3.1 エラーハンドラの設定

あらかじめアプリケーションプログラム内でエラーハンドラを設定しておくと、計算結果に誤りが生じた際にエラー終了せずに、そのエラーハンドラが呼び出されます。エラーハンドラの設定には DS-CUDA の提供する API、`dscudaSetErrorHandler()` を用います。

> **書式**:
> `void dscudaSetErrorHandler(void (*`*handler*`)(void *), void *`*handler_arg*`)`

引数 *handler* に、エラーハンドラへのポインタを渡します。エラーハンドラは `void *` 型の引数をひとつ取れます。この引数を引数 *handler_arg* として与えます。引数が不要の場合には `NULL` を与えてください。

> **参考** : 同一のアプリケーションプログラムを、ソースコードを変更すること無く DS-CUDA プリプロセッサ `dscudacpp` と従来の CUDA コンパイラ `nvcc` の両方で処理できるようにするためには、以下に示すように C プリプロセッサディレクティブを用いて `dscudaSetErrorHandler()` の呼び出しを保護してください。

```
#ifdef __DSCUDA__
    dscudaSetErrorHandler(errhandler, (void *)&data);
#endif
```

ここで `__DSCUDA__` は `dscudacpp` が自動的に定義する定数マクロです (節 4.1 参照)。

4.3.2 自動再計算

DS-CUDA には自動再計算機能が実装されています。つまり、計算結果に誤りが生じた場合にその計算を自動的に再実行させることが出来ます。GPU に恒久的な故障が生じた場合には再実行は無意味ですが、確率的にまれに生じる計算の誤りであれば、再実行によって訂正できる場合があります。

自動再計算機能を用いるには環境変数 `DSCUDA_AUTOVERB` を定義してください。変数は定義さえされていれば、値はどのような値を設定しても構いません。

ただし自動再計算は任意のアプリケーションプログラムで正しく機能するわけではありません。自動再計算機能は以下の手順で再計算を行います。

(1) アプリケーションプログラムの実行中は、CUDA API の呼び出しが行われるたびにその呼び出し履歴 (つまり API 名とすべての引数) を内部バッファに保存します (ただし現在のところ保存するのはホスト-GPU 間のデータ転送を行う CUDA API とカーネル実行のみ)。

(2a) GPU **の計算結果をデバイスからホストへの** `cudaMemcpy()` **によるデータ転送 (以降** D2H **と表記します) によってホストへ回収する際に、回収結果が正しければ過去の呼び出し履歴を消去します。**

(2b) **回収結果が誤っていれば過去の呼び出し履歴を順に再実行します。**

自動再計算が正しく機能するためには、この手順で再計算を行った時に正しい結果を返すようにアプリケーションプログラムが記述されている必要があります。つまり GPU **上の計算結果が、直近の** D2H **から次の** D2H **までの間の** CUDA API **呼び出しだけに依存していることが必要です。**

例えば GPU **上の特定の変数を毎ステップインクリメントさせるプログラムでは、自動再計算は正しく動作しません。再計算を行った時に、その変数が余分にインクリメントされてしまいます。**

5 DS-CUDA 実装の詳細

5.1 CUDA ランタイムライブラリのサポート範囲

DS-CUDA は CUDA ランタイムライブラリのすべての機能や API を仮想化するわけではありません。以下の条件に該当するものを含むいくつかの機能および API は、現在のところ仮想化されていません。

- グラフィクス制御を伴う API。
  例: `cudaGraphicsGLRegisterBuffer()`
- カーネル実行に関する API。
  例: `cudaLaunch()`
- 非同期 API。
  例: `cudaMemcpyAsynch()`
- GPU 間のデータ転送。
  例: `cudaMemcpy(..., cudaMemcpyDeviceToDevice)`, `cudaDeviceEnablePeerAccess()`
- Unified Virtual Addressing を前提としたメモリ操作 API。
  例: `cudaMemcpy(..., cudaMemcpyDefault)`
- GPUDirect 関連機能

5.2 CUDA C/C++ 文法のサポート範囲

DS-CUDA は CUDA C/C++ コンパイラでコンパイル可能なすべてのソースコードを扱えるわけではありません。現在のところ、アプリケーションプログラムのソースコードが以下の条件に該当する記述を含む場合には、そのソースコードは処理できません。

- CUDA カーネルを関数ポインタ経由で呼び出す記法。

例)
```
void (*kernel)(...);

__global__ void myKernel(...)
{
  ....
}
...
void main(void)
{
  kernel = myKernel;
  ...
  kernel<<<grid, threads>>>(...);
}
```

5.3 InfiniBand Verb インタフェース

DS-CUDA の InfiniBand Verb インタフェースは OFED の提供する C 言語ライブラリ関数を用いて記述されています。ライブラリの詳細については OFED の提供するドキュメントを参照して下さい。

5.4 RPC インタフェース

DS-CUDA の RPC インタフェースは XDR 言語を用いて`$dscudapkg/src/dscudarpc.x`内に記述されています。この記述を `rpcgen` によってコンパイルすると、クライアントスタブ `$dscudapkg/src/dscudarpc_clnt.c`、サーバスタブ `$dscudapkg/src/dscudarpc_svc.c` などが生成されます。XDR や `rpcgen` の詳細については別途資料をあたって下さい。

5.5 Client node から server node への CUDA カーネルの転送

`.cu` ファイル内で定義された CUDA カーネル関数は、`dscudacpp` によって抽出され、`dscudacpp` はこれを オプションスイッチ `--ptx` とともに `nvcc` へ渡し、`.ptx` 形式 (高レベルアセンブリ記述) へと変換します。

client node 上のアプリケーションプログラムがカーネル呼び出しを実行すると、client node から上記の `.ptx` ファイルがカーネルイメージとして server node 上のサーバプログラム `dscudasvr` へ転送されます。サーバプログラムは CUDA ドライバ API のひとつ、`cuModuleLoadData()` を使用してこのイメージを GPU へロードし、`cuModuleGetFunction()` を使用してカーネル関数を取り出します。

カーネル関数への引数は別途アプリケーションプログラムから転送されます。サーバは `cuParamSetv()`, `cuParamSeti()`, `cuParamSetf()`, `cuParamSetSize()` を使用して

これらの引数をカーネル関数のコンテクストにセットします。その後、ブロックの情報を `cuFuncSetBlockShape()` によって設定し、`cuLaunchGrid()` によってカーネル関数を起動します。

以上の動作は `$dscudapkg/src/dscudasvr.cu` 内の `dscudaLaunchKernel()` に記述されています。

6 利用許諾

DS-CUDA ソフトウェアパッケージの利用条件についてはファイル `00license-j` をご確認下さい。

7 DS-CUDA ソフトウェアパッケージ更新履歴

version	date	description
1.2.2	06-Sep-2012	一般公開向けに利用許諾等を整備。
1.2.1	09-Aug-2012	初版作成。