Remote CUDA ソフトウェアパッケージ ユーザガイド

for Remote CUDA version 2.0.0

最終更新：2011 年 2 月 26 日

K&F Computing Research Co.

株式会社 K & F Computing Research
E-mail: support@kfcr.jp

目 次

1　本文書の概要

この文書では Remote CUDA ソフトウェアパッケージの使用方法を説明します。Remote 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 GPGPU としてシームレスに使用するためのソフトウェア環境です。本バージョンは CUDA バージョン 3.2 に対応しています。

以降、第 2 章では Remote CUDA の基本構成と動作概要を説明します。第 3 章では Remote 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 章では拡張機能である冗長計算機能について説明します。第 6 章は実効性能の測定値を示します。第 7 章では Remote CUDA の実装や内部動作について触れます。

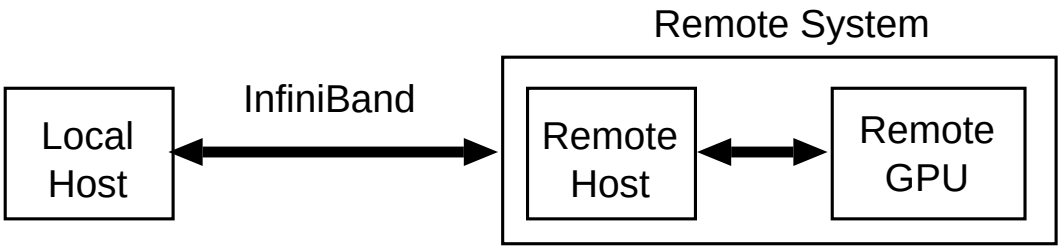
なお以降では、本パッケージのルートディレクトリ (/パッケージを展開したディレクトリ/rcudapkg バージョン番号/) を $rcudapkg と表記します。

2　Remote CUDA 概観

本節では Remote CUDA の基本構成と機能を概観します。

2.1　ハードウェアの構成

Remote CUDA を使用する典型的なシステム構成の例として、下図に示すシステムを考えます。特に記載の無い限り、以降の説明はこのシステムを対象とするものとします。



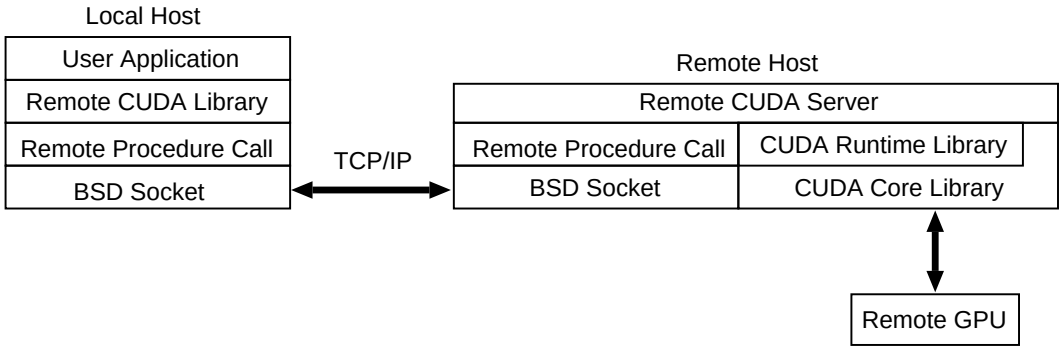
図：Remote CUDA を利用するシステムの例。

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 から成ります。簡便のため、一方の PC をローカルホストと呼び、他方をリモートホストと呼ぶことにします。ネットワーク接続には原則として InfiniBand (10Gb/s × 2 ポート) を想定します。ただし TCP/IP による通信が可能な接続であれば、必ずしも InfiniBand である必要はありません。

リモートホストには 1 枚の GPU カードが接続されているものとし、これを リモート GPU と呼ぶことにします。リモート GPU は NVIDIA 社製の CUDA に対応した製品であることが必須です。

2.2 ソフトウェアの構成

Remote CUDA はユーザに対し、サーバ・クライアント型の実行環境を提供します。すなわち、リモートホスト上ではサーバプログラム `rcudasrv` を常時稼働させておき、ローカルホスト上のユーザプログラムを `rcudasrv` 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ユーザプログラムの要求に従ってリモート GPU を制御します。

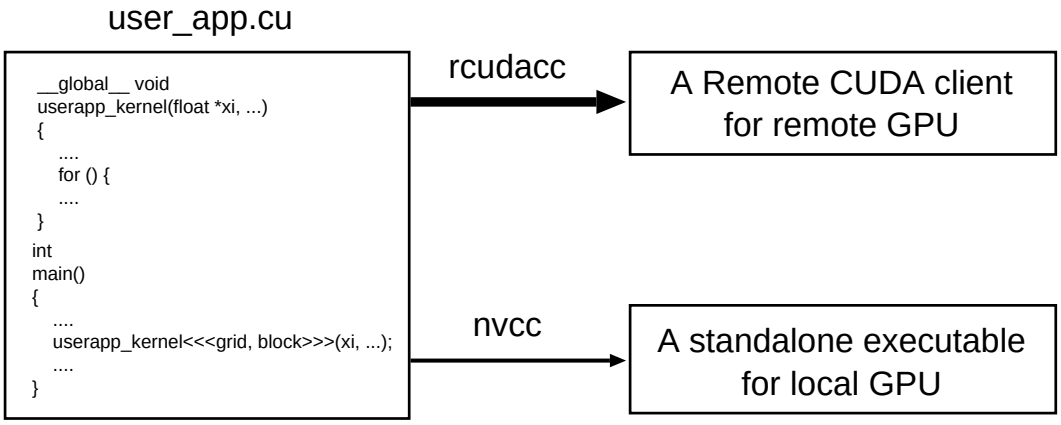


図：ソフトウェア構成

クライアント・サーバ間の通信プロトコルには TCP/IP を用います。通信 API には BSD Socket を使用します。BSD Socket のラッパーとして Remote Procedure Call (RPC) を、さらにその上位のラッパーとして本パッケージの提供する Remote CUDA ライブラリ を使用します。サーバによる GPU の制御は、NVIDIA 社の提供する CUDA Runtime API (一部 CUDA Driver API) を介して行われます。

2.3 クライアントプログラムの生成

CUDA を使用するユーザアプリケーション (つまりリモートホストではなくローカルホストに接続された GPU を使用するアプリケーション) のソースコードを、本パッケージの提供する Remote CUDA コンパイラ `rcudacc` を用いてコンパイルすることにより、Remote CUDA クライアントが生成されます。つまりユーザは、リモートホストに接続された GPU を使用する場合にも、ローカル向けに記述したソースコードをそのまま使用できます。



図：`rcudacc` はローカル GPU 向けのソースコードからリモート GPU 向けのクライアントを生成する。

2.4 冗長計算機能

本章では簡単のため説明を省きましたが、Remote CUDA は冗長計算機能をサポートしています。つまり、複数のリモート GPU 上で同一の計算を実行し、両者の結果が異なっていた場合には、その旨をユーザアプリケーションに通知することが可能です。この機能の使用方法については第 5 章を参照してください。

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 2.2 以降を推奨)
  `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/`

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、Remote 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。

3.2 パッケージの展開

ソフトウェアパッケージ `rcudapkg`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/	
rcudacc	.cu ファイルから Remote CUDA クライアントを生成するコンパイラ。
rcudasvr	Remote CUDA サーバ。
rcudalaunch	Remote CUDA サーバ起動スクリプト。
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。`librcuda.a` が使用します。
rcudatest	Remote CUDA 動作テスト用クライアント。
include/	ヘッダファイル (Remote CUDA クライアント・サーバ共用)。
lib/	
librcuda.a	Remote CUDA ライブラリ。
src/	Remote 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、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。
NVIDIA_GPU_Computing_SDK3.2	Remote CUDA 用 make ファイルを追加した CUDA 3.2 SDK。

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。

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

RCUDA_PATH : Remote 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。

LD_LIBRARY_PATH : 共有ライブラリパスに $RCUDA_PATH/lib を追加してください。設定必須。

RCUDA_SERVER : Remote 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

RCUDA_SERVER_CONF : Remote CUDA サーバの設定ファイル。冗長計算機能を使用する際に、スクリプト$rcudapkg/bin/rcudalaunch が参照します。rcudalaunch を用いない場合には設定は不要です。
デフォルト値は $RCUDA_PATH/misc/server.conf

RCUDA_WARNLEVEL : Remote 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。

例:

```
kawai@localhost>export RCUDA_PATH="/home/kawai/src/rcudapkg2.0.0"
kawai@localhost>export RCUDA_SERVER="192.168.10.101"
kawai@localhost>export LD_LIBRARY_PATH=/home/kawai/src/rcudapkg2.0.0/lib:\
$LD_LIBRARY_PATH
```

これらの他に 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 $rcudapkg/src へ移動し、make を実行してください。Remote CUDA ライブラリ $rcudapkg/lib/librcuda.a と Remote CUDA サーバ $rcudapkg/bin/rcudasvr、

Remote CUDA テストプログラム$rcudapkg/bin/rcudatest が生成されます。

```
kawai@localhost>pwd
/home/kawai/src/rcuda2.0.0/src
kawai@localhost>make
rpcgen -N -l rcudarpc.x > rcudarpc_clnt.c
rpcgen -N -h rcudarpc.x > rcudarpc.h
cp rcudarpc_clnt.c rcudarpc_clnt.cu
...
ranlib librcuda.a
c++ -I. -I/usr/local/cuda3.2/cuda/include -I/home/kawai/src/cuda3.2\
/NVIDIA_GPU_Computing_SDK/C/common/inc -fPIC -shared -o libcudart.so\
.3 cudart_dummy.c /usr/local/cuda3.2/cuda/bin/nvcc  -g -I. -I/usr/lo\
cal/cuda3.2/cuda/include -I/home/kawai/src/cuda3.2/NVIDIA_GPU_Comput\
ing_SDK/C/common/inc -o rcudatest rcudatest.cu -L../lib -lrcuda
kawai@localhost>
```

以上でインストールは完了です。

3.5 動作チェック

テストプログラム $rcudapkg/bin/rcudatest と、$rcudapkg/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 テストプログラム rcudatest

リモートホスト上でサーバ rcudasvr を起動し、

```
[root@localhost]# ./rcudasvr  &
[1] 1234
server id : 0
ndevice : 1
real device       : 0
virtual device    : 0
[root@localhost]#
```

ローカルホスト上でテストプログラムを実行します。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/rcuda2.0.0/src
kawai@localhost>./rcudatest
usage: ./rcudatest <test_program_ID> [destination_IP_address]
   0) shows GPU status.
   1) measure send (local->remote) performance.
   2) measure receive (local<-remote) performance.
   3) measure device-memory write (host->GPU) performance.
   4) measure device-memory read (host<-GPU) performance.
```

引数に 1 を与えて実行すると、ホストから GPU へのデータ転送速度の測定を行います。

```
kawai@localhost>./rcudatest 1

#
# Raw send (local host -> remote host)
#
Client IP address : 192.168.1.2
4096 byte    4.560770 sec    43.852244 MB/s
10240 byte    1.978837 sec    101.069478 MB/s
25600 byte    1.112054 sec    179.847364 MB/s
64000 byte    0.766354 sec    260.975969 MB/s
160000 byte    0.486547 sec    411.059986 MB/s
400000 byte    0.378098 sec    528.963375 MB/s
1000000 byte    0.330105 sec    605.868146 MB/s
2500000 byte    0.300943 sec    664.577903 MB/s
6250000 byte    0.288833 sec    692.441314 MB/s
kawai@localhost>
```

引数に 2 を与えると、GPU からホストへのデータ転送速度の測定を行います。

```
kawai@localhost>./rcudatest 2
#
# Raw receive (local host <- remote host)
#
Client IP address : 192.168.1.2
4096 byte    4.745010 sec    42.149541 MB/s
10240 byte    2.254493 sec    88.711741 MB/s
25600 byte    1.336569 sec    149.636861 MB/s
64000 byte    0.968696 sec    206.463147 MB/s
160000 byte    0.632387 sec    316.262074 MB/s
400000 byte    0.482722 sec    414.317105 MB/s
1000000 byte    0.416788 sec    479.860148 MB/s
2500000 byte    0.424247 sec    471.423457 MB/s
6250000 byte    0.432278 sec    462.665317 MB/s
kawai@localhost>
```

3.5.2 サンプルプログラム

$rcudapkg/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 Remote CUDA クライアントが生成されます。

```
kawai@localhost>pwd
/home/kawai/src/rcuda2.0.0/sample/claret
kawai@localhost>make
cc -DVTGRAPE -O -ffast-math -funroll-loops  -o sockhelp.o -c sockhelp.c
../../bin/rcudacc -o mr3.o -I. -I/usr/local/cuda/include \
  -I/usr/local/cuda/NVIDIA_CUDA_SDK/common/inc -c -use_fast_math \
  -O -i mr3.cu
Info    : verbose:2
Info    : infile:mr3.cu
Info    : ptxfile:mr3.cu.ptx
...
cc -DVTGRAPE -O -ffast-math -funroll-loops  -I/home/kawai/src/cuda3.2/\
NVIDIA_GPU_Computing_SDK/shared/inc cras36.c -o cras_gpu sockhelp.o mr\
3.o /home/kawai/src/cuda3.2/NVIDIA_GPU_Computing_SDK/C/lib/libcutil_x8\
6_64.a -L/usr/local/cuda3.2/cuda/lib64 -L../../lib -lcudart -L/\
usr/local/lib -lglut -lGL -lGLU -lm -lstdc++ -lrcuda
kawai@localhost>
```

3.5.3 CUDA SDK サンプルプログラム

$rcudapkg/NVIDIA_GPU_Computing_SDK3.2/C/src/ 内に、NVIDIA 社の提供する CUDA SDK 3.2 に含まれている各種のサンプルプログラムのうち、グラフィクス関連の CUDA API を用いないものすべてが含まれています (全 56 種)。

各サンプルプログラムのディレクトリ内には、Remote CUDA 用の make ファイルが Makefile.rcuda というファイル名で用意されています。各ディレクトリ内で make -f

Makefile.rcuda を実行すると、それぞれの Remote CUDA クライアントが生成されます。

```
kawai@localhost>pwd
/home/kawai/src/rcuda2.0.0/NVIDIA_GPU_Computing_SDK3.2/C/src/reduction
kawai@localhost>make -f Makefile.rcuda
echo objs reduction_kernel.cu.o reduction.cpp.o
objs reduction_kernel.cu.o reduction.cpp.o
echo src reduction_kernel.cu
src reduction_kernel.cu
/home/kawai/src/rcuda2.0.0/bin/rcudacc -o reduction_kernel.cu.o  --ptxa
...
/usr/local/cuda3.2/cuda/bin/nvcc  --compiler-options -fPIC -m64 -o redu\
ction -L/home/kawai/src/rcuda2.0.0/lib -L/home/kawai/src/cuda3.2/NVIDIA\
_GPU_Computing_SDK/C/lib -L/home/kawai/src/cuda3.2/NVIDIA_GPU_Computing\
_SDK/shared/lib reduction_kernel.cu.o reduction.cpp.o -lrcuda -lshrutil\
_x86_64 -lcutil_x86_64
kawai@localhost>
```

なお全 56 種のうち、以下に示す 8 種のプログラムは Remote CUDA では動作しません。残り 48 種については動作確認済みです。

- transpose : このサンプルプログラム中では、以下に示すように、CUDA カーネルを関数ポインタ経由で呼び出しています。Remote CUDA はこのような記述を扱えません。

```
void (*kernel)(float *, float *, int, int, int);
...
kernel = &copySharedMem;
...
kernel<<<grid, threads>>>(d_odata, d_idata, size_x, size_y, 1);
```

- 下表に示す 7 種のサンプルプログラムは、CUDA Toolkit の提供するライブラリを使用しています。これらのライブラリはその内部からリモート化されていない CUDA カーネルを呼び出しています。そのためこれらのサンプルプログラムを Remote CUDA を用いてコンパイルしても、正しく動作しません。

サンプルプログラム名	依存するライブラリ
conjugateGradient	libcublas.so
convolutionFFT2D	libcufft.so
simpleCUFFT	libcufft.so
MonteCarloCURAND/EstimatePiQ	libcurand
MonteCarloCURAND/EstimatePiP	libcurand
lineOfSight	libcudpp_x86_64
radixSort	libcudpp_x86_64

4 Remote CUDA コンパイラ rcudacc の使用方法

CUDA 拡張 C 言語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 Remote CUDA クライアントを生成するには、Remote CUDA コンパイラ $rcudapkg/bin/rcudacc を使用します。

rcudacc 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/rcuda2.0.0/sample/direct
kawai@localhost>../../bin/rcudacc
No input file given.
  usage: ../../bin/rcudacc [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.
```

rcudacc 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 Remote 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、rcudacc では解釈されずに、rcudacc 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 rcudacc への引数として渡さねばなりません。また、クライアントは Remote CUDA ライブラリを使用するため、引数 -lrcuda を与えてこれをリンクする必要があります。

例えば、本パッケージ付属のサンプルプログラム $rcudapkg/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。

```
kawai@localhost>pwd
/home/kawai/src/rcuda2.0.0/sample/vecadd
kawai@localhost>../../bin/rcudacc -o userapp -I. -i userapp.cu -lrcuda
```

参考：rcudacc 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。

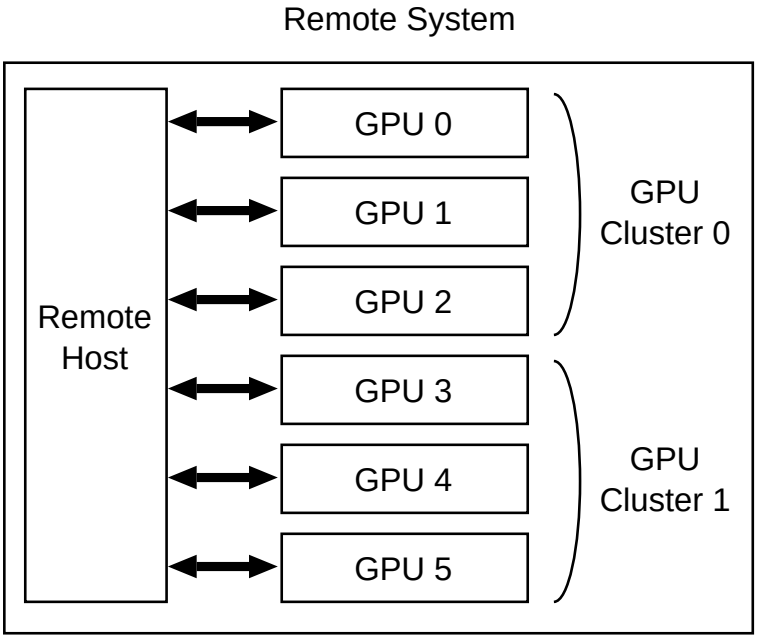
定数名	値
__RCUDA__	1
__RCUDACC_VERSION__	バージョン番号 (例：バージョン 1.2.3 の値は 0x010203)

5　冗長計算機能 の使用方法

Remote CUDA は冗長計算機能をサポートしています。つまり、複数のリモート GPU 上で同一の計算を実行し、両者の結果が異なっていた場合には、その旨をユーザアプリケーションに通知することが可能です。この章では冗長計算機能の使用方法を説明します。

5.1　リモートホスト側の設定

説明に用いるリモートシステムの例として、下図に示すものを考えます。



図：複数の GPU を持つリモートシステムの例。

Remote CUDA はリモートホスト上に搭載された複数の GPU をいくつかのグループに分割して管理します。このグループのことを「GPU クラスタ」と呼ぶことにします。そして各 GPU クラスタにはユニークな非負整数値「GPU クラスタ ID」を割り当てるこにします。

GPU クラスタと Remote CUDA サーバは 1 対 1 に対応します。つまり、ひとつの GPU クラスタは、ひとつの Remote CUDA サーバによって制御されます。ひとつのリモートホストに複数の GPU クラスタが接続されている場合、そのリモートホスト上には複数の Remote CUDA サーバを稼働させることが可能です。

上図のシステムでは、ひとつのリモートホストに 6 つの GPU が接続されています。例えばこれらの GPU を 3 枚ずつの GPU クラスタにまとめ、それぞれを別個の Remote CUDA サーバに管理させられます。

各 Remote CUDA サーバへの GPU の割り当ては、Remote CUDA サーバの起動時にコマンドライン引数で次のように指定します:

```
rcudasvr -c cluster_id -d 'device_id_list'
```

ここで *cluster_id* は Remote CUDA サーバが管理する GPU クラスタの ID です。任意のユニークな非負整数値を指定できます。*device_id_list* はその GPU クラスタに属する GPU のデバイス番号を、空白で区切って並べたリストです。デバイス番号とは、各 GPU に CUDA が割り当てる整数値です。リモートホストに n 個の GPU が接続されている場合、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。

上図のように 6 枚の GPU を 3 枚ずつ 2 つの GPU クラスタにまとめて管理するには、2 つの Remote CUDA サーバを次のように起動します。

```
[root@localhost]# ./rcudasvr -c 0 -d '0 1 2' &
[1] 1234
server id : 0
ndevice : 3
real device       : 0 1 2
virtual device    : 0 1 2
[root@localhost]# ./rcudasvr -c 1 -d '3 4 5' &
[2] 1235
server id : 1
ndevice : 3
real device       : 3 4 5
virtual device    : 0 1 2
```

クラスタ構成をあらかじめ設定ファイルに記述しておけば、Remote CUDA サーバ起動時の操作を簡略化できます。設定ファイルの書式は次の通りです。

cluster_id : *device_id_list*
cluster_id : *device_id_list*
...

例えば前述の例の GPU クラスタ構成は、以下のように記述します
(C++ 形式のコメントを挿入できます)。

```
// clusterID : deviceID0 deviceID1 ...
0 : 0 1 2  // the 1st cluster
1 : 3 4 5  // the 2nd cluster
```

設定ファイル名を環境変数 RCUDA_SERVER_CONF で指定し、$rcudapkg/bin/rcudalaunch を実行すると、設定にしたがって rcudasvr が起動されます。

```
[root@localhost]# export RCUDA_SERVER_CONF="./server.conf"
[root@localhost]# cat server.conf
// clusterID : deviceID0 deviceID1 ...
0 : 0 1 2  // the 1st cluster
1 : 3 4 5  // the 2nd cluster
[root@localhost]# ./rcudalaunch
RCUDA server configuration : [{:svrid=>"0", :devids=>["0 1 2"]},\
{:svrid=>"1", :devids=>["3 4 5"]}]
./rcudasvr -c 0 -d '0 1 2' &
./rcudasvr -c 1 -d '3 4 5' &
server id : 0
ndevice : 3
real devices      : 0 1 2
virtual devices   : 0 1 2

server id : 1
ndevice : 3
real devices      : 3 4 5
virtual devices   : 0 1 2
[root@localhost]#
```

5.2 ローカルホスト側の設定

ローカルホスト上のクライアントから特定のサーバ、つまり特定の GPU クラスタへアクセスするには、リモートホストの IP アドレスと GPU クラスタ ID の組を環境変数 RCUDA_SERVER に指定します:

書式:
$ip\_address$: $cluster\_id$

ローカルホストから個々の GPU へアクセスするには、実際のデバイス ID ではなく、Remote CUDA によって設定される仮想的なデバイス ID を用います。指定した GPU クラスタに n 個の GPU が属している場合、これらの GPU には 0 から $n-1$ までの仮想デバイス ID が割り当てられます。

例:
```
kawai@localhost>export RCUDA_SERVER="192.168.10.101:0"
```

空白で区切って複数の GPU クラスタを指定すると、それらのクラスタを用いて冗長計

算が行われます (最大 4 個のクラスタを指定できます)。

例:
```
kawai@localhost>export RCUDA_SERVER="192.168.10.101:0 192.168.10.101:1"
```

つまり各 GPU クラスタ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ローカルホスト上で検証されます。一致しなかった場合にはエラーハンドラが呼び出されます。エラーハンドラはアプリケーションプログラム内であらかじめ設定しておきます。設定には RCUDA の提供する API、`rcudaSetErrorHandler()` を用います。

書式:
`void rcudaSetErrorHandler(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 Remote CUDA と通常の CUDA の両方でコンパイル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 `rcudaSetErrorHandler()` の呼び出しを保護してください。

```
#ifdef __RCUDA__
    rcudaSetErrorHandler(errhandler, (void *)&data);
#endif
```

ここで `__RCUDA__` は `rcudacc` が自動的に定義する定数マクロです (第 4 章を参照)。

6 性能実測

Remote CUDA ライブラリと、それを使用した Remote CUDA クライアントの実測性能を以下に示します。測定に使用したリモート GPU は GeForce GTX280 です。

6.1 Remote CUDA ライブラリの通信性能

データ長 (byte)	ローカル ⇒ リモート		ローカル ⇐ リモート	
	InfiniBand	100Base	InfiniBand	100Base
				(MB/s)
128	1.8	0.3	1.8	0.5
281	3.9	0.7	3.9	0.7
619	8.4	1.4	8.4	2.0
1362	17.9	2.7	17.7	2.7
2998	36.4	3.7	35.1	3.7
6596	71.5	6.2	67.6	6.2
14512	123.2	8.0	113.1	8.0
31927	185.1	9.6	164.7	9.6
70241	247.0	10.1	197.3	10.6
154530	386.1	11.2	305.9	11.2
339967	509.3	11.5	396.4	11.5
747927	590.1	11.5	466.7	11.5
1645440	639.9	11.0	493.0	10.9
3619968	667.1	10.8	453.8	10.8
7963931	678.4	9.8	452.7	9.8

6.2 アプリケーションプログラム `claret` の実効性能

	改良版			オリジナル		
粒子数	ローカル	リモート		ローカル	リモート	
		InfiniBand	100Base		InfiniBand	100Base
						(Gflops)
8	0	0	0	0	0	0
64	1	0	0	1	0	0
216	6	3	1	6	2	0
512	16	10	3	13	6	1
1000	33	24	8	29	17	4
1728	57	46	18	52	35	11
2744	89	75	33	84	63	22
4096	130	110	52	125	99	39
5832	174	150	77	170	136	60

注意：オリジナル版の claret は、タイムステップ毎にデバイスメモリの確保 `cudaMalloc()` と解放 `cudaFree()` を行っていますが、これらは粒子数が変わらない限りは不要です。改良版はメモリの確保と解放を必要最小限に抑えたものです。

7 Remote CUDA 実装の詳細

7.1 Remote CUDA ライブラリのサポート範囲

Remote CUDA ライブラリは CUDA ランタイムライブラリのすべての API をリモート化するわけではありません。以下の条件に該当する API は、現在のところリモート化されていません。

- 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()`)

7.2 Remote CUDA コンパイラのサポート範囲

Remote CUDA コンパイラは CUDA 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コンパイルでき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コンパイルできません。

- 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>>>(...);
}
```

7.3 RPC インタフェース

Remote CUDA の RPC インタフェースは XDR 言語を用いて`$rcudapkg/src/rcudarpc.x`内に記述されています。この記述を `rpcgen` によってコンパイルすると、クライアントスタブ `$rcudapkg/src/rcudarpc_clnt.c`、サーバスタブ `$rcudapkg/src/rcudarpc_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。

7.4 リモートホストへの 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、rcudacc によって抽出され、rcudacc 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。

Remote CUDA クライアント実行時に、クライアントは上記の .ptx ファイルをカーネルイメージとしてリモートホスト上のサーバ rcudasvr へ転送します。サーバは CUDA ドライバ API のひとつ、cuModuleLoadData() を使用してこのイメージをロードし、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クライアントから転送されます。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 $rcudapkg/src/rcudasvt.cu 内の rcudalaunchkernel_1_svc() に記述されています。

8 Remote CUDA ソフトウェアパッケージ更新履歴

version	date	description	author(s)
2.0.0	26-Feb-2011	CUDA3.2 の API に対応 (グラフィクス関連 API ほか一部を除く)。 冗長化機能を実装。 C++ テンプレートに対応。	AK
1.0.1	26-Oct-2009	rcudatest に機能 3,4 を追加。	AK
1.0.0	22-Oct-2009	初版作成。	A. Kawai

お問い合わせ:
株式会社 K&F Computing Research (support@kfcr.jp)