ハイブリッド計算機のプログラミング環境RIVER

理化学研究所
情報基盤センター

野田　茂穂

RICCとGPGPU

RICC
RIKEN Integrated Cluster of Clusters

RICC設計コンセプト

I. 次世代スーパーコンピュータを活用するアプリケーションの開発環境

II.新しい計算機技術の利用

III.加速器・次世代シーケンサなどの実験データのストレージ・解析

数千並列規模の並列ジョブ実行が可能

１００ノードにグラフィックボードを利用したアクセラレータを搭載

4PBのテープアーカイブ装置と500TBのディスク装置が実験結果の大規模データに対応

RICC
RIKEN Integrated Cluster of Clusters

RICC設計コンセプト

I. 次世代スーパーコンピュータを活用するアプリケーションの開発環境

II.新しい計算機技術の利用

III.加速器・次世代シーケンサなどの実験データのストレージ・解析

数千並列規模の並列ジョブ実行が可能

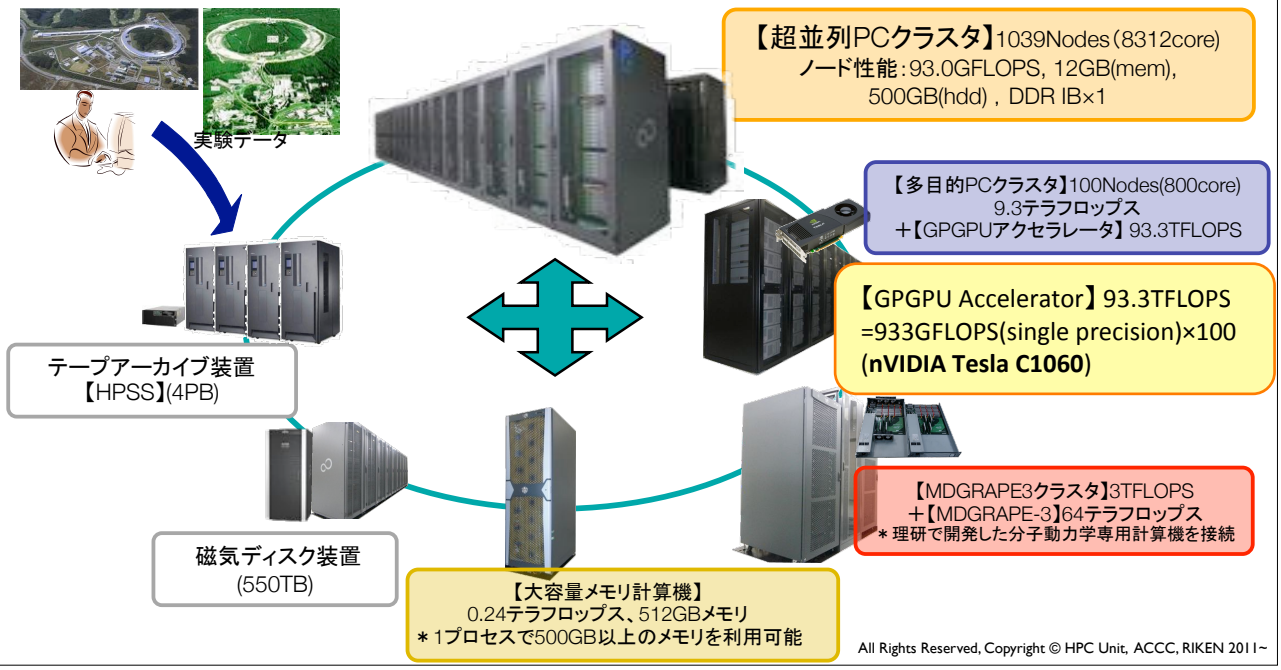
１００ノードにグラフィックボードを利用したアクセラレータを搭載

4PBのテープアーカイブ装置と500TBのディスク装置が実験結果の大規模データに対応

All Rights Reserved, Copyright © HPC Unit, ACCC, RIKEN 2011~

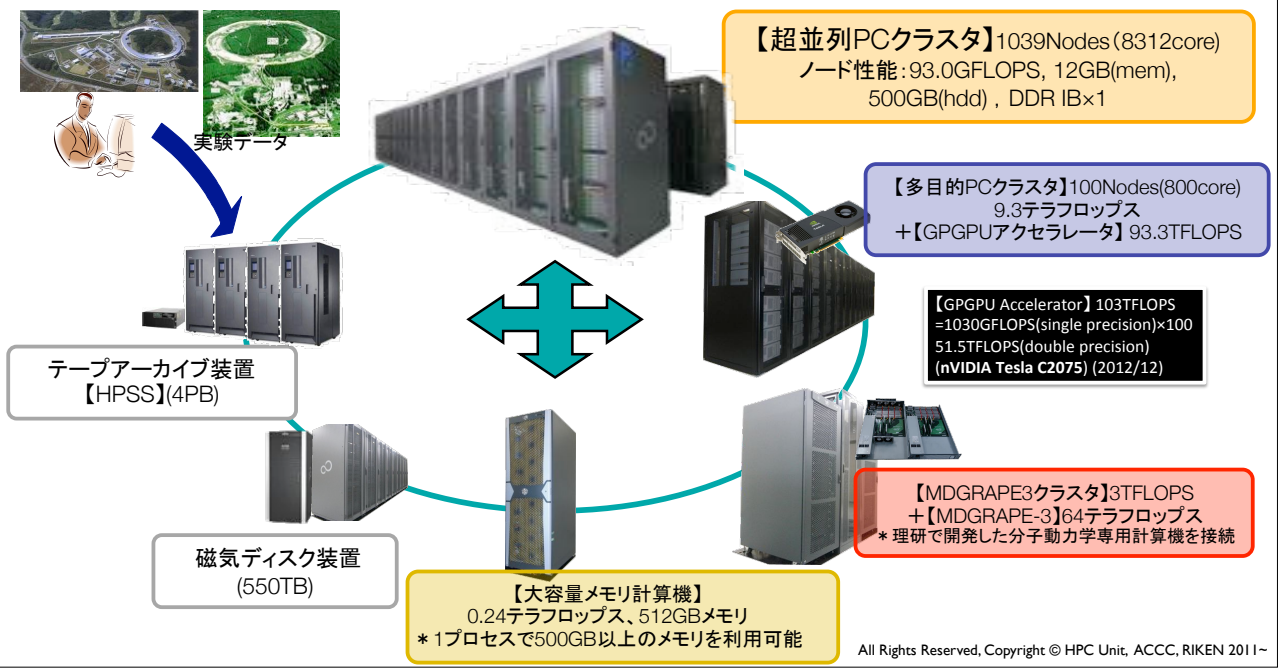
RICCの概要(2009~)

【システム構成】
超並列PCクラスタ＋GPUクラスタ＋専用機クラスタ＋大容量メモリ計算機



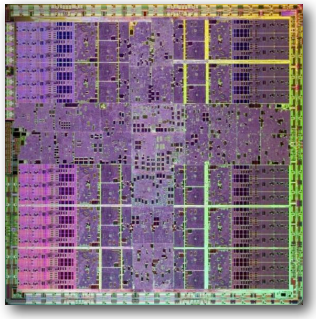
RICCの概要(2009~)

【システム構成】
超並列PCクラスタ＋GPUクラスタ＋専用機クラスタ＋大容量メモリ計算機



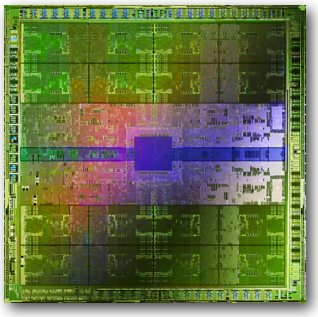
C1060(65m) vs. C2075 (40nm)[アーキテクチャ]

GT200 Generation GPU
（Tesla T10 series : C1060）



1.4 billion transistors
240 cores (= shaders)
30 cores for Double Precision

"Fermi" Generation GPU
（Tesla T20 series : C2075）



3.0 billion transistors
512 cores (= shaders)
256 cores for Double Precision

C1060 vs. C2075 – Key Specifications

	C1060 	C2075 
Single Precision	933 GFlops	1030 GFlops
Double Precision	78 GFlops	515 GFlops
Memory	4 GB GDDR3	6 GB GDDR5
Memory Bandwidth	102 GB/s	144 GB/s
Architecture	GT200	Fermi

C1060 vs. C2070 – Hardware Architecture

	GT200 (C1060)	Fermi (C2075)
Transistors	1.4 billion	3.0 billion
CUDA Cores	240	448
Floating Point Operations	IEEE 754	IEEE 754-2008
- Double Precision Floating Point	30 FMA ops/clk	256 FMA ops/clk
- Single Precision Floating Point	240 MAD ops/clk	512 FMA ops/clk
Special Function Units (per SM)	2	4
Warp schedulers (per SM)	1	2
Shared Memory (per SM)	16 KB	Configurable 48/16 KB
L1 Cache (per SM)	-	Configurable 16/48 KB
L2 Cache	-	768 KB
ECC Memory Support	-	Yes
Concurrent Kernels	-	Up to 16
Load/Store Address Width	32-bit	64-bit

C1060 vs. C2070 – Software, System management

	C1060	C2075
Display output	No	1x Dual Link DVI on C2075
GPUDirect v2.0 (Peer- to-Peer communication)	No	Yes
Unified Virtual Addressing	No	Yes
NVIDIA Management Library		
- Board Serial Number query	No	Yes
- Power reading (current draw, etc.)	No	Yes
CUDA support		
- CUDA 3.x	Yes	Yes
- CUDA 4.x	Yes	Yes
- CUDA 5.x	No	Yes

インストールされているアプリ／ライブラリ

アプリケーション一覧

Name	ricc	mpc	upc	mdg	ax	Install directory
ADF			○			/usr/local/adf2008.01
ADF			○			/usr/local/adf2009.01
AMBER 8				○		/usr/local/amber8
AMBER 10			○			/usr/local/amber10
AMBER 11			○			/usr/local/amber11
ANSYS	○		○			/usr/local/ansys
DALTON		○	○			/usr/local/dalton
DMol3		○				/usr/local/Accelrys
GAMESS		○	○			/usr/local/gamess
Gaussian03			○			/usr/local/gaussian/g03_E01
Gaussian03NBO			○			/usr/local/gaussian_nbo/g03_E01
Gaussian09			○			/usr/local/gaussian/g09.A.02
GaussView	○					/usr/local/gaussian/gv
GROMACS		○	○			/usr/local/gromacs-4.5.3
GRRM			○			/usr/local/GRRM-1.020
molden	○					/usr/local/molden4.7
meep		○	○			/usr/local/meep-1.1
NAMD		○	○			/usr/local/namd-2.7b1
VMD	○					/usr/local/vmd-1.8.7
Hermes		○				/usr/local/GOLD_Suite
ROOT	○	○	○			/usr/local/root-5.26-00a
VisIt	○					/usr/local/VisIt
Smoldyn		○	○			/usr/local/smoldyn-2.12

ライブラリ一覧

Name	ricc	mpc	upc	mdg	ax	Install Directory
IMSL	○	○	○			/opt/vni/imsl/fnl600/lnxin100e64
IMSL	○	○	○			/opt/vni/imsl/fnl600/lnxin110e64
GSL	○	○	○	○		/usr/local/lib
GSL	○	○	○	○	○	/usr/local/lib
GSL	○	○	○	○		/usr/local/lib
FFTW3	○	○	○	○	○	/usr/local/lib
FFTW2	○	○	○	○	○	/usr/local/lib
FFTW2-MPI	○	○	○	○	○	/usr/local/lib
HDF5	○	○	○	○	○	/usr/local/lib
ATLAS	○	○	○	○		/usr/local/atlas3.9.3
Harminv		○	○			/usr/local/lib
libctl		○	○			/usr/local/lib
Xerces-C++	○	○	○	○		/usr/local/lib
libxml2	○	○	○	○		/usr/local/lib

RICCポータル
https://ricc.riken.jp

Python-2.4 ライブラリ

Name	ricc	mpc	upc	mdg	ax	Install Directory
nose	○	○	○	○		/usr/lib64/python2.4/site-packages
numpy	○	○	○	○		/usr/lib64/python2.4/site-packages
scipy	○	○	○	○		/usr/lib64/python2.4/site-packages
Rodis		○	○	○		/usr/lib64/python2.4/site-packages
mpi4py	○	○	○	○		/usr/lib64/python2.4/site-packages
Matplotlib(Py Lab)	○	○	○	○		/usr/lib64/python2.4/site-packages
pytz	○	○	○	○		/usr/lib64/python2.4/site-packages
dateutil	○	○	○	○		/usr/lib64/python2.4/site-packages

Python-2.6.5 ライブラリ

Name	ricc	mpc	upc	mdg	ax	Install Directory
nose	○	○	○	○		/usr/local/python-2.6.5/lib/python2.6/site-
numpy	○	○	○	○		/usr/local/python-2.6.5/lib/python2.6/site-
scipy	○	○	○	○		/usr/local/python-2.6.5/lib/python2.6/site-
mpi4py		○	○			/usr/local/python-2.6.5/lib/python2.6/site-

RICC
RIKEN Integrated Cluster of Clusters

RIVER (RIKEN IBM Visual Programing Environment)

CPU, GPGPU混在環境でのプログラミング環境

メニーコア化・ハイブリッド化の流れ

- CPUの多コア化、および多量のコアを備えたアクセラレータの混載により、HPCのピーク性能は大幅に向上した
  - すでにハイブリッド・システムが、Top 500の上位を多数占めている
- さらなるメニーコア化、ハイブリッド化の進行が予想される

RICC

Xeon 4 **コア×2** + GPGPU 240**コア**
=248core/node

Tianhe-2

Xeon Ivy 12 **コア×2** + Xeon Phi 57**コア×3**
=195core/node

TOP10 June 2013

1. **Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P
   NUDT
2. **Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x
   Cray Inc.
3. **Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom
   IBM
4. K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect
   Fujitsu
5. **Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom
   IBM
6. **Stampede** - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P
   Dell
7. **JUQUEEN** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect
   IBM
8. **Vulcan** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect
   IBM
9. **SuperMUC** - iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR
   IBM
10. **Tianhe-1A** - NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050
    NUDT

直面するプログラミングの困難

- メニーコア化、ハイブリッド化が進むとプログラミングが課題になる

タスク並列化

アーキテクチャごとの最適化

- 処理AはCPUで、処理BはGPUで・・・
- キャッシュサイズ、命令セットの違いを意識した最適化・・・

通信の実装

- CPU同士はMPIで、GPUとは専用の転送で・・・
- CPUの演算とGPUの演算と通信は同時に実行・・・

計算リソースへの割り当て

- 処理AはCPUでN並列、処理BはGPUだからM並列で・・・

性能と生産性を両立させる新しい枠組みが必要

直面するプログラミングの困難

・メニーコア化、ハイブリッド化が進むとプログラミングが課題になる

タスク並列化

アーキテクチャごとの最適化

・処理AはCPUで、処理BはGPUで・・・
・キャッシュサイズ、命令セットの違いを意識した最適化・・・

？？

通信の実装

・CPU同士はMPIで、GPUとは専用の転送で・・・
・CPUの演算とGPUの演算と通信は同時に実行・・・

計算リソースへの割り当て

・処理AはCPUでN並列、処理BはGPUだからM並列で・・・

複雑化

”通信の実装”と”計算リソースへの割当”をアシスト

Riken Ibm Visual programing EnviRonment
理研 & 日本IBM東京基礎研究所

ハイブリッド・システムプログラミング環境 “RIVER”

2レベルプログラミング

タスクの自動並列化

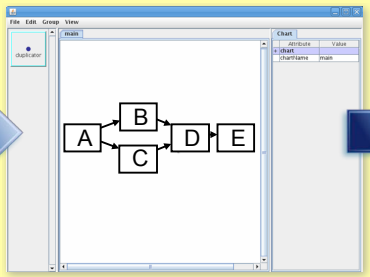
コンポーネント・プログラミング

A.c　D.cpp　C.c　B.f95　C.cu　E.c

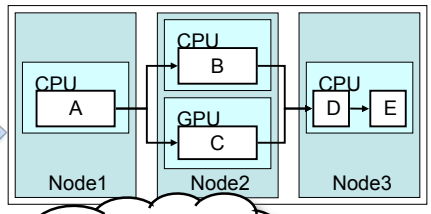
タスク並列化

アーキテクチャごとの最適化

データ・フロー・プログラミング



コンパイル



通信の実装

計算リソースへの割り当て

熟練プログラマーにより作成したものを共有
シングルスレッド用に書かれた既存の言語・資産を流用可能
追加はわずかな指示文だけ

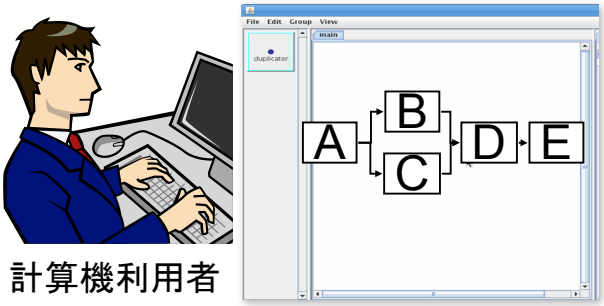
一般プログラマーは、アルゴリズムのデータフローを記述するだけ
最適化された部品ライブラリと（左）、利用するシステムに最適化された並列化（右）を自動的に利用できる

コンパイラーが、プロセッサーごとの処理能力とシステム構成を考慮して処理の割り当てと並列度を判断
通信コードも自動的に挿入

RIVERを利用したプログラミングのイメージ

1. アルゴリズムのデータフローを記述する

(1) 基礎式に対する数値解析用の離散化アルゴリズムを設計
(2) 離散化アルゴリズムをデータフロー図として記述
(3) シリアルプログラムコーディング
(4) 並列プログラムコーディング



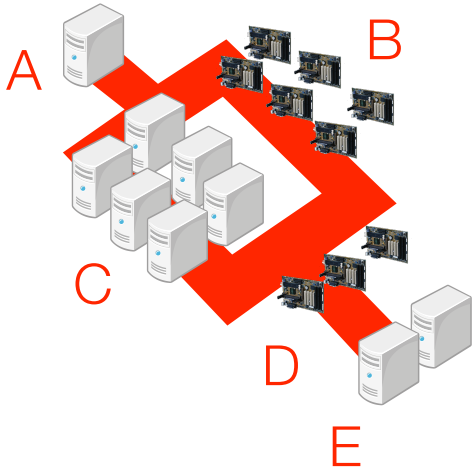
計算機利用者

2. システム構成を与えてコンパイル

3. コンパイラーが、プログラムを生成

処理内容とコードライブラリを参照して、最適な計算ノード、並列数を決定。

必要なプログラム部品、通信コードを含んだ実行形式を、ノードごとに生成。(MPMD)



RIVERソフトウェアスタックと開発状況

これまでの実現してきたこと

アプリケーション層	ZZ-EFSI	PhaseField	BlackSholes	分子動力学アプリ	•••
拡張オペレータ層	境界条件UDOP	初期値生成UDOP	•••		
基本オペレータ層	四則演算UDOP	微分演算UDOP	積分演算UDOP	密行列演算	•••
RIVERコンパイラ・ランタイム層	RIVER並列化コンパイラ・ランタイムライブラリ				
ネイティブコンパイラ・ランタイム層	OpenMPI/MPICH	•••			
	gcc/gfortran, icc/ifort, pgc/pgfortran, nvcc	•••			

基本オペレータを拡充することで、構築可能なアプリケーションのカバレッジを拡大できます

プログラミング例

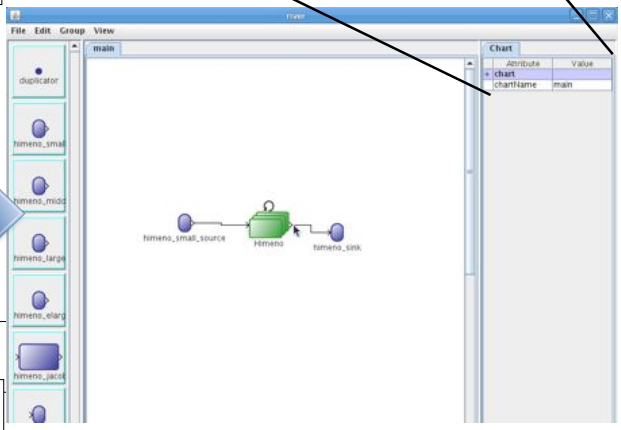
```
/// KERNEL himeno_sink
/// > HIMENO_SINK_UDOP

/// KERNEL himeno_small_source
/// > HIMENO_SMALL_SOURCE_UDOP
void himeno_small_source(float

/// KERNEL jacobi
/// > HIMENO_JACOBI
void jacobi(float *in, float *out)

/// KERNEL jacobi_cuda
/// > HIMENO_JACOBI
/// ARCH x86
/// ACCEL cuda
void jacobi_cuda(float *in, float
*out) {
....
}
```

Block	Port
Attribute	Value
+ group	
class	Himeno
loop	800
parallel	8
template	sleeve
image	



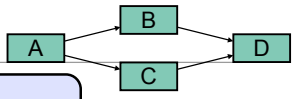
```
/// UDOP HIMENO_SMALL_SOURCE_UDOP
/// +float[129][65][65] in

/// UDOP HIMENO_SINK
/// +float[][][] in

/// UDOP HIMENO_JACOBI_UDOP
/// +float[][][] in
/// -float[][][] out
```


リソース自動割当

Target Stream Program



Step0. 最適化表の事前作成

Target UDOPs: A, B, C, D

UDOP	Kernel	Pattern	Pitch	Resource
A	A1	loop(1, A_x86)		x86 x 1
B	B1	loop(1, B_x86)		x86 x 1
B	B2	loop(1, B_cu)		cuda x 1
C	C1	loop(1, C_x86)		x86 x 1
D	D1	loop(3, D_x86)		x86 x 1
D	D2	splitjoin(3, D_x86)		x86 x 3
D	D3	loop(3, D_cu)		cuda x 1
D	D4	splitjoin(3, D_cu)		cuda x 3
D	D5	splitjoin(loop(1, D_x86), loop(2, D_cu))		x86 x 1, cuda x 1

Step 1. 実行パターン仮選択

最適化表から最小パイプラインピッチを持つ実行パターンを仮選択

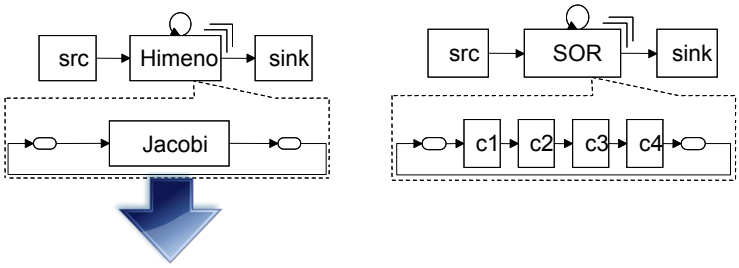
Step 2. 計算リソース制約を解く

より計算リソースを消費しない実行パターンに切り替える

Step 3. 通信経路選択

通信コストの大きいエッジを速い通信路にマッピング。現在は、ポインター渡し, メモリーコピー, ソケット通信（TCP/IP）,MPI(Infiniband) から選択。

部品ライブラリ（姫野ベンチ,SOR4）での性能実験



HimenoBMT (M size = 128 x 128 x 256) での割り当て例

実験環境

	x86	GPGPU
言語	C	C
コンパイラー	icc	nvcc
並列度	1-80	1-80
コア数またはカード数	1-80コア	1-80カード
使用ノード数	10	80
特記事項	8コア/ノー	CUDA for C

ホスト数	スケジューリング結果
1	GPGPU x 1
2	GPGPU x 1
4	GPGPU x 4
8	GPGPU x 8
16	GPGPU x 16
32	GPGPU x 16
64	GPGPU x 16
80	GPGPU x 16

2ホストは通信コストが高く1並列で実行

ホストを増やすとスケールする領域

16ホスト以上は頭うちと判断

実験結果

HimenoBMT (L size = 256 x 256 x 512)

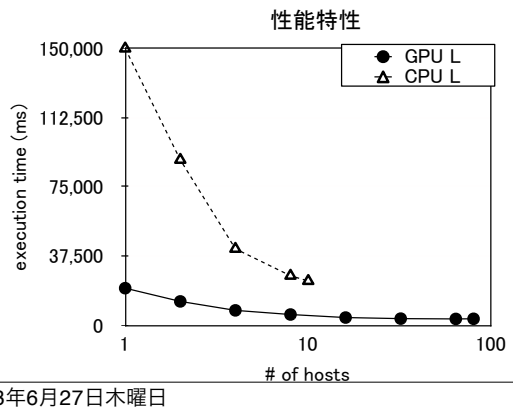
SORBMT (L size = 512 x 512 x 512)

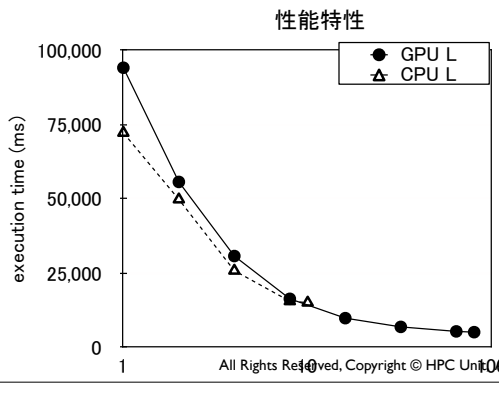
自動並列化・スケジューリング結果

使用可能ホスト数	スケジューリング結果
1	GPGPU x 1
2	GPGPU x 2
4	GPGPU x 4
8	GPGPU x 8
16	GPGPU x 16
32	GPGPU x 32
64	GPGPU x 64
80	GPGPU x 80

スケジューリングミス；最速構成は GPGPU x 64

使用可能ホスト数	スケジューリング結果
1	x86 x 8 コア
2	x86 x 16コア
4	x86 x 32コア
8	x86 x 64コア
16	GPGPU x 16
32	GPGPU x 32
64	GPGPU x 64
80	GPGPU x 80

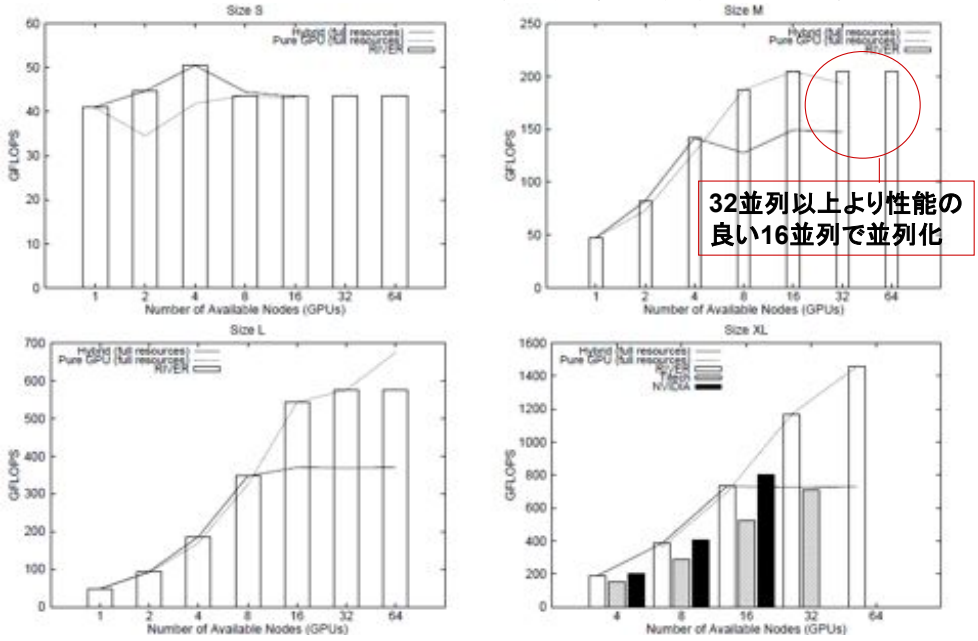




RIVERによる自動リソーススケジューリングと並列化効率

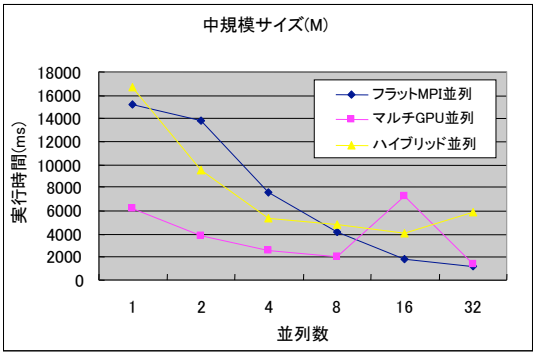
姫野ベンチマーク性能評価

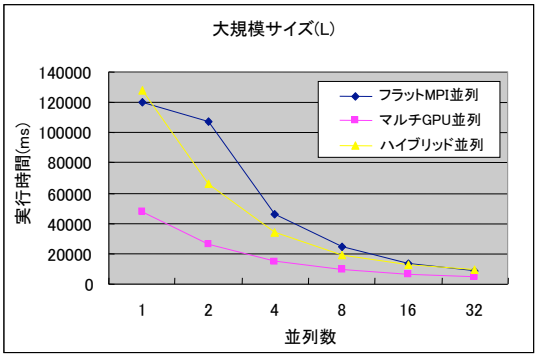
実線：指定された並列数で並列化
RIVER：性能劣化する場合は、指定された並列数以下で並列化



GPUを用いたHimenoBMTの性能評価

- RICCにおける実行時間と並列化効率

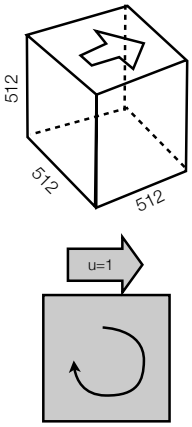


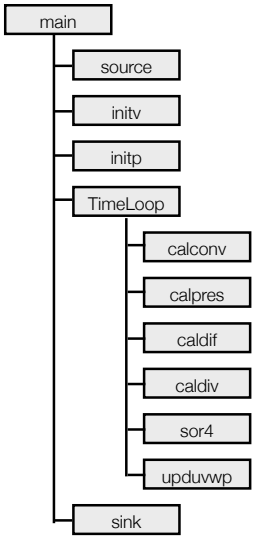


- 問題規模 中規模サイズ(M): 258x258x258、大規模サイズ(L): 514x514x514
- フラットMPI並列：MPIのみでの並列化
- ハイブリッド並列：ノード内はスレッド並列＋MPI通信スレッド（通信隠蔽）、ノード間はMPI通信。
- マルチGPU並列：複数のGPUを使用。ノード間通信はCPUで動作（通信隠蔽）

アプリケーションの構築（流体解析の場合）

- 非圧縮流体解析のプログラム（Cavity問題, SMAC法）をRIVERで構築
- 元のコードはFortranで記述された非並列のコード
- グローバル変数は使用していない





File Name	Line No.
main.f	20
source.f	83
initv.f	34
initp.f	32
calconv.f	95
calpres.f	68
caldif.f	76
caldiv.f	49
sor4.f	361
updduvwp.f	68
sink.f	24

RIVERへの移植（半日程度）

- サブルーチン毎のUDOP化
  - サブルーチンヘッダにKERNELの記述

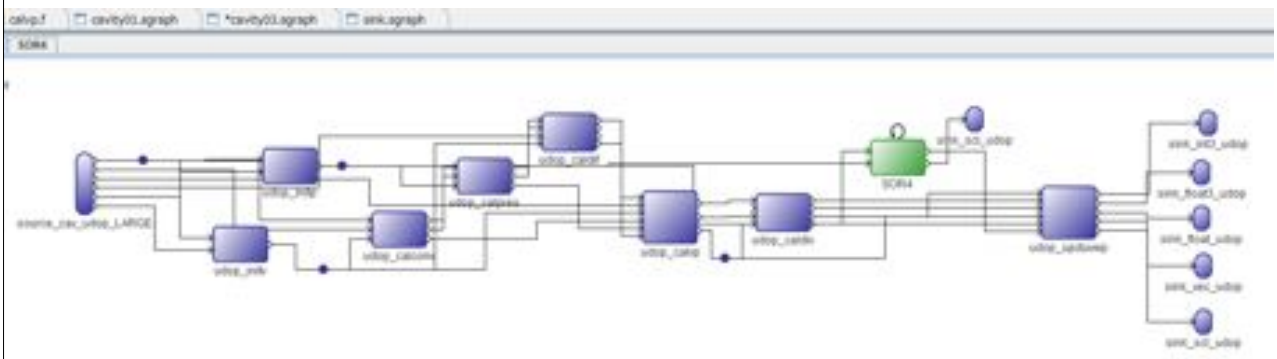
```
!/// KERNEL calconv(index,csize,uvw,uvwconv)
!/// > udop_calconv
!/// ARCH x86
      subroutine calconv(index,csize,uvw,uvwconv)
```

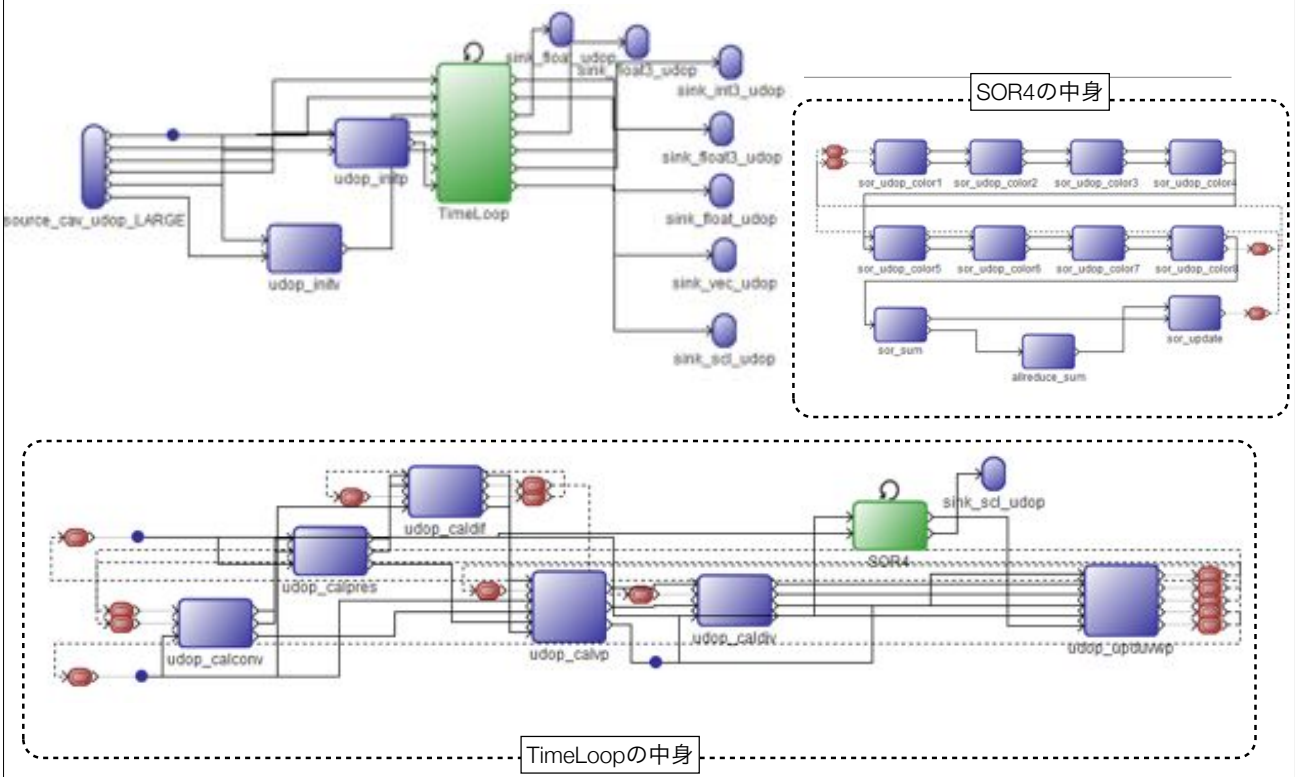
  - UDOP定義のファイルを作成
    - 変数の型、IN/OUT
    - 変数領域内の依存関係

```
!/// UDOP udop_calconv
!/// +-int[3] index
!/// +-float[3] csize
!/// +float[][][][3] uvw
!/// -float[][][][3] uvwconv
!/// DEP uvwconv->uvw [1,0,0,0;0,1,0,0;0,0,1,0;0,0,0,1]
[0,0,0,0;0,0,-1,0;0,-1,0,0;-1,0,0,0;1,0,0,0;0,1,0,0;0,0,1,0]
    :
    :
```

File Name	Line No.	ΔLine No.
main.f	20	不要
source.f	83	3
initv.f	34	3
initp.f	32	3
calconv.f	95	3
calpres.f	68	3
caldif.g	76	3
caldiv.f	49	3
sor4.f	361	ライブラリ使用
updduvwp.f	68	3
sink.f	24	6
udop.def		92

ビジュアルエディターでのプログラム作成（半日）



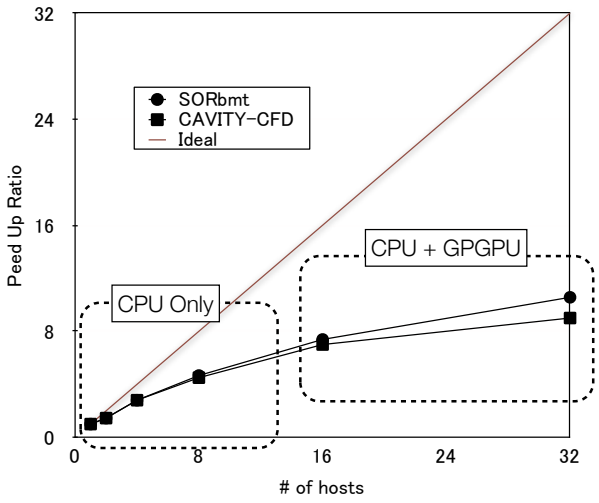


並列性能

- SORbmt : Loop=100
- CAVITY-CFD : TimeLoop=1000, SORLoop=1000

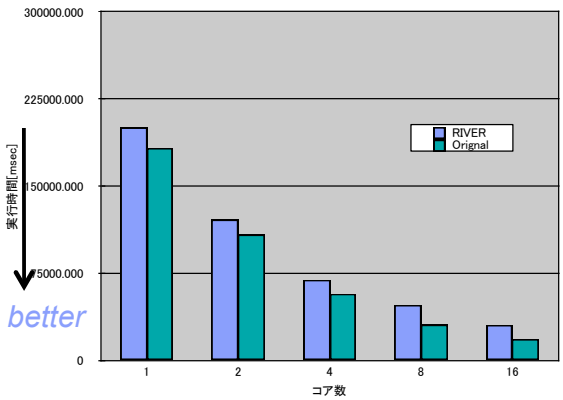
SORBMT (L size = 512 x 512 x 512)

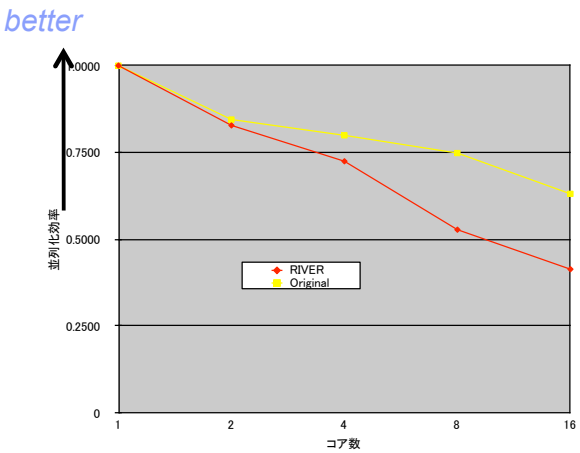
使用可能ホスト数	スケジューリング結果
1	x86 x 8 コア
2	x86 x 16コア
4	x86 x 32コア
8	x86 x 64コア
16	GPGPU x 16
32	GPGPU x 32



ZZ-EFSI(流体構造連成シミュレーション)の性能評価

- RICCにおける実行時間と並列化効率





- 問題規模：格子サイズは70x70x70(境界を含む）、時間発展は100ステップ
- 使用ブロック数：74種類の基本UDOPの組み合わせ、合計232ブロック

まとめ

- 複雑な計算機環境向けのソフトウェア開発環境として、以下の機能をもつ開発環境を構築
  - 通信コードの自動埋め込み
  - 計算リソースへの自動割当
- これにより、非並列プログラムから並列プログラムを生成
- CPU, GPU混在環境にも対応
- 人がチューニングした性能には至らないが、並列性能は悪くない

- 興味のある方は、shigeho@riken.jp　まで。

＃土居、村瀬、前田、小松、野田、姫野. MPI-OpenCLハイブリッド並列を実現するプログラミング・フレームワークの設計と実装. 情報処理学会

＃SACSIS 2012 M. Murase, K. Maeda, M. Doi, H. Komatsu, S. Noda, and R. Himeno. Automatic Resource Scheduling with Latency Hiding for Parallel Stencil Applications on GPGPU Clusters. In Proceedings of the IEEE IPDPS 2012,, 2012.

＃M. Murase, K. Maeda, M. Doi, H. Komatsu, S. Noda, and R. Himeno. A parallel programming framework orchestrating multiple languages and architectures. In Proceedings of the 8th ACM International Conference on Computing Frontiers, CF '11, 2011. to be published.