2014年度HPC Challenge Awards Competition Class 2に向けて

中尾 昌広

理化学研究所 計算科学研究機構

HPC Challenge(HPCC) Benchmarks

- HPCシステムを多角的に評価するためのベンチマークセット
  - MPIによるリファレンス実装が公開されている（http://www.hpcchallenge.org）
- HPCC Award Competition at Supercomputer Conference
  - In Class 1, only the performance of an HPC system is evaluated
  - In Class 2, the productivity and performance of a programming language are evaluated（コードのエレガントさを50%、性能を50%で評価）
    - STREAM
    - RandomAccess
    - High Performance Linpack (HPL)
    - Fast Fourier Transform (FFT)

- 4つのHPCC Benchmarksの内3つ以上を選択
- 2つまでの，他のベンチマークを追加可能

2013年度の結果

- 昨年は京スパコンを利用：4つのHPCCベンチマーク + 姫野ベンチマーク
- 最優秀賞を受賞（日本初）

- 昨年のレポートはXMPの公式ページ（http://xcalablemp.org）からダウンロードできます

2014年度に向けて（1/2）

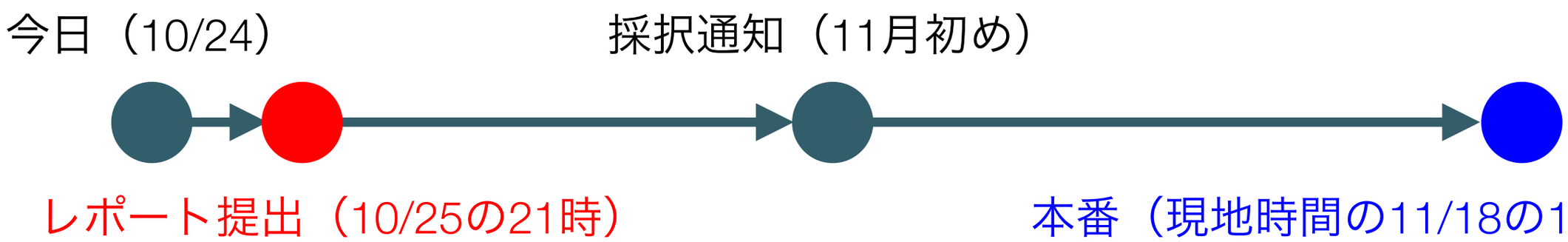
- XcalableACCを用いてHPCCベンチマークを実装
- HPL，FFT，STREAM（RandomAccessは性能が出ないと予測されるのでパス）
  - しかし，HPLはチューニング不十分，FFTは未実装
- 追加ベンチマークとして姫野ベンチマーク（ステンシルアプリケーション）
- HA-PACS/TCAで評価（16ノード以上を使うので、TCAは利用せず）

http://www.ccs.tsukuba.ac.jp

CPU	Ivy Bridge E5-2680v2 2.8 GHz x 2
Memory	DDR3 128 GByte, 59.7 GB/s
Network	Infiniband 4xQDR x 2 rails, 8GB/s
GPU	K20X, 1.31/3.95 TFlops（DP/SP）

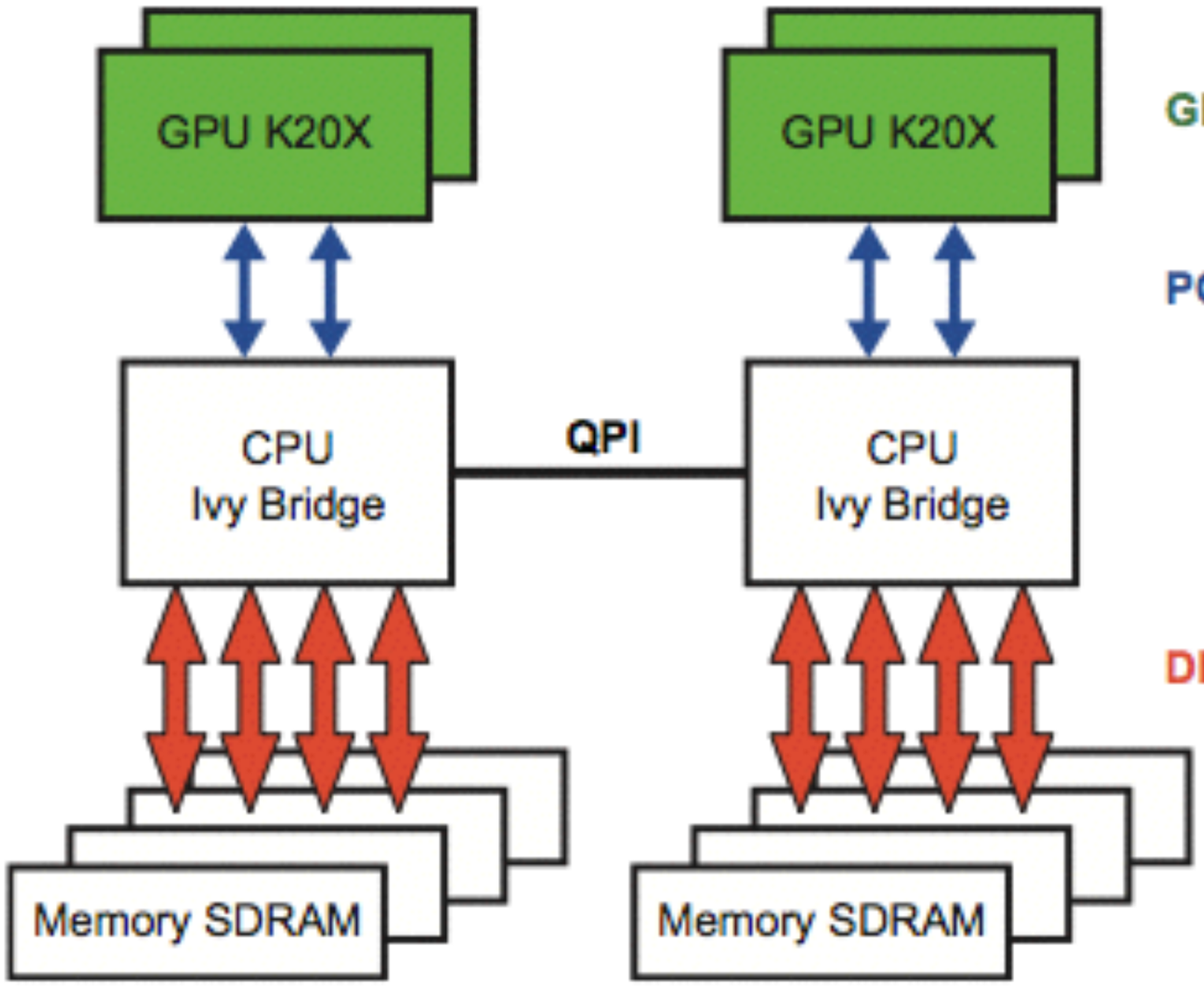
2014年度に向けて（2/2）

- 注意：今回の結果は色々と初期評価です



- 毎年10/24がレポートの締め切り．神戸に帰る新幹線の中で完成予定
- FinalistはSC@ニューオリンズのBoFでプレゼン発表（毎年2〜4件）
- 本番までに値を更新してもよいので、それまで頑張ります

HA-PACS/TCAのノードスペック



- 1ノードにつき4プロセスを実行
- 各プロセスは1つのGPUを操作

STREAM：実装

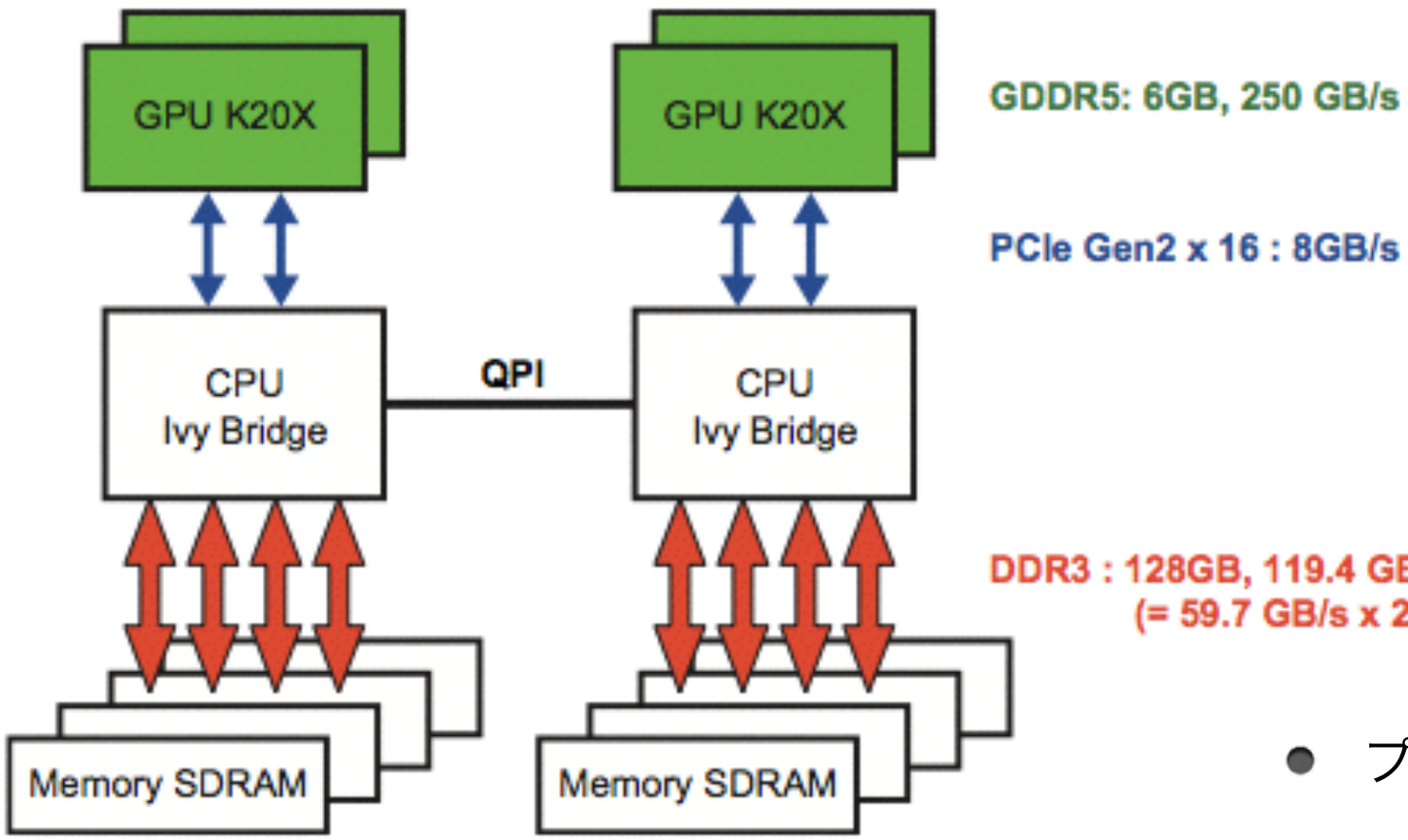
- メモリバンド幅を主に測定
- 非常に単純なベンチマーク. **カーネル部分**に通信は発生しない

```
for(k=0; k<NTIMES; k++) {
    times[k] = -xmp_wtime();

  for (j=0; j<size; j++)
    a[j] = b[j] + scalar*c[j];

  times[k] += xmp_wtime();
}
```

- 条件：システムメモリの1/4以上を a[], b[], c[]の配列は使うこと
- ただし、システムメモリはアクセラレータのメモリは入らない

STREAM実装

- 条件：システムメモリの1/4以上を a[], b[], c[]の配列は使うこと
- ただし、システムメモリはアクセラレータのメモリは入らない



- プロセスあたりのメモリは32GB
- その1/4は8GByte
- しかし、GPUメモリは6GByteしかない

STREAM実装

- GPUとCPUの同時利用
- GPUに入りきらない分は、CPUで処理する（GPU-CPU間の通信は行わない）
- OpenACCで試したところ、6.0GByteの内、5.5GByteくらいは確保できた
- つまり、GPUは5.5GByteに対し、CPUは2.5GByteを処理する
- GPUの方が圧倒的に速いため、CPUが処理している間にGPUが終わる
- ノードのピークバンド幅は 250GB/s x 4 + 59.7GB/s x 2 = 約1.1TB/s
- しかし CPU律速なので、実際の限界は 59.7GB/s x 2 x 8 / 2.5 = 約380GB/s

STREAM実装

```
#pragma xmp nodes p(*)
#pragma acc data copy(a[0:GPU_SIZE], b[0:GPU_SIZE], c[0:GPU_SIZE])
  {
    for(k=0; k<NTIMES; k++) {
#pragma xmp barrier
   times[k] = -xmp_wtime();

#pragma acc parallel loop async
      for (j=0; j<GPU_SIZE; j++)
        a[j] = b[j] + scalar*c[j];

#pragma omp parallel for
      for (j=GPU_SIZE; j<MAX_SIZE; j++)
        a[j] = b[j] + scalar*c[j];

#pragma acc wait

#pragma xmp barrier
   times[k] += xmp_wtime();
   }
} // acc data
```

GPUが処理

CPUが処理

GPUの処理が終わるのを待つ

STREAMの結果

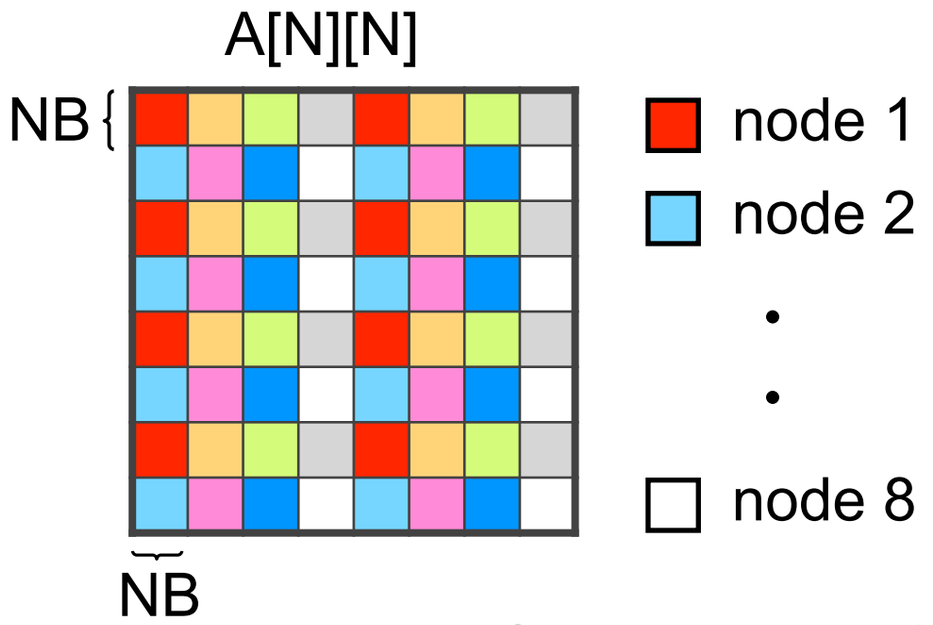
#Nodes	#CPUs	#GPUs	Performance (GB/s)	
			XcalableACC (/peak)	XcalableMP (/peak)
1	1	1	**112.98 (20.2%)**	36.16 (6.5%)
1	2	4	**226.96 (20.3%)**	72.74 (6.5%)
2	4	8	**453.71 (20.3%)**	145.18 (6.5%)
4	8	16	**906.61 (20.3%)**	289.99 (6.5%)
8	16	32	**1,812.30 (20.2%)**	580.23 (6.5%)
16	32	64	**3,623.14 (20.2%)**	1,159.52 (6.5%)
32	64	128	**7,238.10 (20.2%)**	2,318.59 (6.5%)

- 括弧内のピーク性能は 約1.1TB/s/nodeで計算
- 59.7GB/s x 2 x 8 / 2.5 = 約380GB/s の値から考えると、60%程度なので妥当
- XMP版のSTREAMは69行、XACC版のSTREAMは84行

High Performance Linpack (HPL)

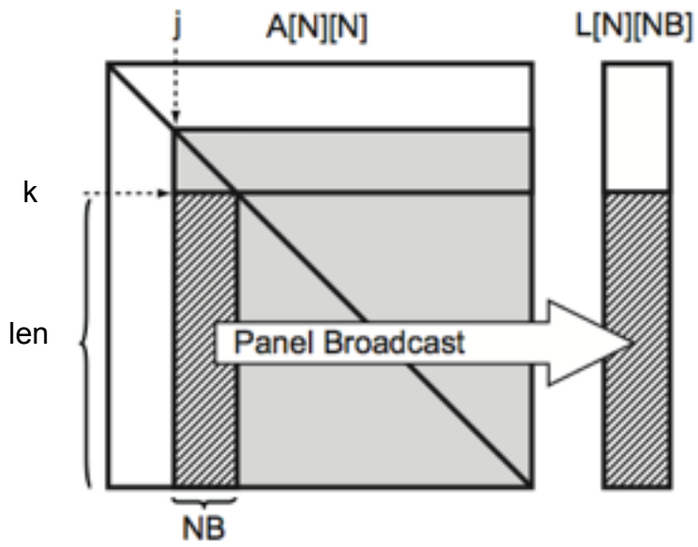
- 密行列の連立一次方程式をLU分解で解く
  - 係数行列を2次元ブロックサイクリック分割する（hpcc-1.4と同じ）

```
double A[N][N];
#pragma xmp nodes p(4,2)
#pragma xmp template t(0:N-1, 0:N-1)
#pragma xmp distribute t(cyclic(NB), \
                         cyclic(NB)) onto p
#pragma xmp align A[i][j] with t(j,i)
```



- **gmove**指示文を用いたパネル転送

```
double A_L[N][NB];
#pragma xmp align L[i][*] with t(*,i)
     :
#pragma xmp gmove
L[k:len][0:NB] = A[k:len][j:NB];
```



CUBLASの利用

- **cuBLASの利用**

```
#pragma acc data copyin(devA_L[0:local_len_y], devA_U[0:local_len_x])
for(int n=0;n<local_len_y;n+=NB){
  // devAに必要なデータをパックして保存
#pragma acc data copy(devA[0:local_len_x])
{
#pragma acc host_data use_device (devA_L, devA_U, devA)
    cublasDgemm('n','n', local_len_x, NB, NB, -1.0, devA_U, local_len_x,
                         devA_L+n*NB, NB, 1.0, devA, local_len_x);
```

- OpenACCのdata指示文を使って、アクセラレータにデータを送信
- OpenACCのhost_data指示文を使って、cublasDgemmにアクセラレータのポインタを渡して実行
- XMP版のHPLは313行、XACC版のHPLは343行

HPLの結果

- ピークのたった5%
- ホストとGPU間の
  データの送受信に要する
  時間が多いから
- 実は性能が低いことは
  わかっていた

#Nodes	#CPUs	#GPUs	Performance **GFlops (/peak)**
1	1	1	**91.14 (5.94%)**
1	2	4	**284.54 (5.00%)**
2	4	8	**520.52 (4.76%)**
4	8	16	**1,021.81 (4.77%)**
8	16	32	**1,870.62 (4.42%)**
16	32	64	**3,757.59 (4.46%)**
32	64	128	**7,102.36 (4.22%)**

- GPUの処理中にデータをパイプライン的に送らないと性能は出ない。
  本実装は、その前段階として作成したもの
- パイプラインに書いたものも作成済だが、OpenACCコンパイラの
  不具合のため、明日が〆切のレポートには間に合わず
- 本番で発表できれば、更新後の値を公開する

姫野ベンチマーク

- 非圧縮流体解析コードの性能評価のためにポアッソン方程式解法を
  ヤコビの反復法で解く場合に主要なループの処理速度を計るもの
- ステンシル計算
- TCAネットワークを利用した方が速いが、16ノード以上を用いたかっ
  たので、今回はInfiniband（MPI）を使った実装
  - 16ノード以上も通信可能なTCAとMPIのハイブリッド通信ライブラリを
    筑波大で開発中

姫野ベンチマーク

- 姫野ベンチマークの実装

```
float p[MIMAX][MJMAX][MKMAX];
// XMP指示文を用いて分散配列の定義
#pragma xmp shadow p[1:1][1:1][0]

#pragma acc data copy(p) ..
{
..
#pragma xmp reflect (p) acc
..
#pragma xmp loop (k,j,i) on t(k,j,i)
#pragma acc parallel loop ..
for(i=1 ; i<MIMAX ; ++i)
  for(j=1 ; j<MJMAX ; ++j){
#pragma acc loop vector ..
    for(k=1 ; k<MKMAX ; ++k){
      S0 = p[i+1][j][k] * ..;
```

袖領域の定義

アクセラレータに分散配列を転送

アクセラレータメモリの袖交換

ループの分散処理

逐次版姫野ベンチマークに
対して，指示文を挿入するのみ

姫野ベンチマーク

#Nodes	#CPUs	#GPUs	Performance (GFlops)		
			XcalableACC (/peak)	OpenACC+MPI (/peak)	MPI (/peak)
1	1	1	**56.27 (1.35%)**	56.81 (1.38%)	13.81 (0.17%)
1	1	2	**111.09 (1.37%)**	112.40 (1.38%)	
1	2	4	**218.73 (1.35%)**	222.72 (1.37%)	27.04 (0.17%)
2	4	8	**437.15 (1.35%)**	444.20 (1.37%)	53.29 (0.16%)
4	8	16	**868.74 (1.34%)**	886.95 (1.36%)	105.72 (0.16%)
8	16	32	**1,734.18 (1.33%)**	1,768.40 (1.36%)	208.80 (0.16%)
16	32	64	**3,466.15 (1.33%)**	3,515.03 (1.35%)	414.18 (0.16%)
32	64	128	**6,870.98 (1.32%)**	6,897.32 (1.33%)	820.22 (0.16%)

- XACC版の姫野ベンチマークは213行
  - 逐次版の姫野ベンチマークに指示文を挿入するのみで作成可能
- 既存のMPI版の姫野ベンチマーク（325行）と、それをOpenACC化したもの（365行）との比較
- XACCとMPI+OpenACCの性能はほぼ同じ

他のPGAS言語との行数の比較

- Comparison with Other PGAS Implementations & MPI (hpcc-1.4)
  - HPCC Awards class2 in 2011 and 2012

Lang.	STREAM	HPL	FFT
XACC	84	343	300くらい？
Coarray Fortran	63	786	450
Chapel	72	658	(NO DATA)
X10	60	708	236
MPI (hpcc-1.4)	329	8,800	1904

まとめ

- HPCC2014への投稿内容を紹介
- STREAMと姫野ベンチマークは、性能を保ったまま簡潔に記述可能
- HPLの性能はあまり良くない
- FFTも作成半ば
- もし採択されたら、SC14のBoFで発表します